Epiphany System Guide

epiphany
e

Clarity and Relevance 3.2

July 1998

CONTENTS

INTRODUCTION

Welcome to Clarity and Relevance 3.2, Epiphany's Enterprise Relationship Management (ERM) suite of decision-making tools that give you quick access to detailed data in simple, straightforward terms.

Clarity provides a window into the data that exists throughout the enterprise—leads, customers, orders, production, and financial reporting. Using Clarity reports and charts, you can analyze complex data, focusing on the aspects that interest you.

Relevance helps you to quickly understand your current data and enables you to forecast quarterly projections and future trends. Using a Web browser, can view all of this data numerically or graphically.

About This Guide

The *Epiphany System Guide* is intended for database administrators and consultants who will install and configure the Clarity and Relevance 3.2 software and will be responsible for maintaining an Epiphany site.

This *Guide* consists of the following chapters and appendices:

Chapter 1, *Basic Concepts*, describes the Epiphany database schema and introduces Epiphany-related data warehousing concepts. Additional basic concepts are presented as special topics in Chapters 2 and 3.

Chapter 2, *Epiphany Database Extraction*, describes the Epiphany database extraction process.

Chapter 3, *EpiCenter Manager*, explains how to set up your organization's data warehouse, which is called an EpiCenter.

Chapter 4, *Web Builder and Ticksheets*, is a guide to Web Builder that explains how to create and modify *ticksheets*. At the front end, users open a ticksheet in a Web browser and select options for the kind of data they want to display. (A *ticksheet* is form that allows end users to construct queries; it is called a ticksheet because users select, or tick, items on a page.)

Chapter 5, *The Epiphany Application Server*, is an operator's guide to the component of the Epiphany ERM application suite that processes all user requests and returns query data in HTML format.

Appendix A, *Installation*, describes the Epiphany system prerequisites and provides instructions for installing and configuring Windows NT 4.0, Microsoft SQLServer, and the Epiphany Clarity and Relevance 3.2 software. Instructions for setting up your Epiphany Application Server are also given.

Appendix B, *Epiphany Macros*, defines the Epiphany-supplied system calls and SQL macros.

Appendix C, *EpiCenter Configuration*, describes the configuration data for a default EpiCenter.

Appendix D, *Date Dimension Fields*, defines the date dimension fields used by the Epiphany system.

Appendix E, *Physical Type Values*, defines the database type translations for the physical types used by the Epiphany system.

Appendix F, *Writing Staging SQL Statements*, explains how to write these SQL statements for base dimension tables and fact tables. Instructions for using external tables as inputs to staging queries are also given.

Appendix G, *Semantic Types*, describes the dimension and fact semantic types.

Appendix H, *Export/Import of Metadata*, presents an overview of the Epiphany system's use of metadata as a basis for a discussion of how the Epiphany export/import metadata feature works.

Appendix I, *Troubleshooting*, describes the Epiphany error conditions and error messages and suggests corrective action.

The *Glossary* defines the terms used throughout this Guide.

How To Use This Guide

A suggested approach to using this *Guide* follows:

- See Appendix A, *Installation*, for instructions on installing and configuring the software and setting up your system.

- Read Chapter 1, *Basic Concepts*, and Chapter 2, *Epiphany Database Extraction*, for the background material you need to set up an EpiCenter. Refer to the *Glossary* for definitions of terms.

- Follow the instructions in Chapter 3, *EpiCenter Manager*, to configure your EpiCenter.

- Run the EpiChannel program, as explained in Chapter 2, *Epiphany Database Extraction*, to extract data from your source systems and place it into the Epiphany tables.

- Start the Application Server as described in Chapter 5, *Epiphany Application Server*, and verify that it is working.

- Construct ticksheets for your organization as described in Chapter 4, *Web Builder and Ticksheets*.

- Refer to the appendices for more detailed information as directed throughout this *Guide*.

CHAPTER ONE

Basic Concepts

This chapter introduces the Epiphany database schema and provides an overview of the Epiphany system.

Epiphany Database Schema

The Epiphany database schema is based on the dimensional data warehouse model. A data warehouse transforms the raw data from an organization's source system databases into data accessible for query and analysis. The data conforms to the organization's business model and is consistent, reusable, and flexible (that is, the data may be re-sorted by any measure the business uses). A data warehouse also provides the tools needed to query, analyze, and publish this data.[1]

The dimensional data warehouse organizes data in what may be visualized as a cube. One can figuratively slice along any dimension of the cube of data to obtain information about the intersection of the dimensions at that point. For example, if the cube has the dimensions Customer, Territory, and Time, and one selects order amount as the measure, one could slice through the cube on the Time dimension to determine the total order amount by customer or territory at a specific date.

---

[1] Ralph Kimball. *The Data Warehouse Toolkit.* John Wiley & Sons, Inc. 1996. [This is an excellent book about data warehousing.]
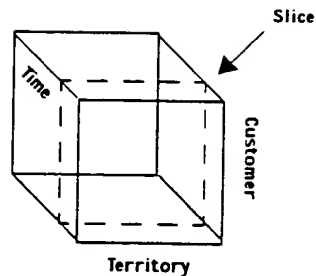
In the Epiphany system, an organization's data warehouse is known as an *EpiCenter*. (An organization may have multiple EpiCenters.)
The Epiphany Application Suite consists of "front-end" Web-based applications, such as Clarity™ and Relevance™, which are designed for database query and analysis. These applications allow users to query the EpiCenter by selecting dimensions and facts they want to know more about. The results of these queries (shown in reports, charts, and graphs within the user's browser window) are derived from the intersection of these dimensions.

The EpiCenter represents a dimensional data warehouse with database tables organized in a *star schema* (see Figure 1-1, page 3). At the center of a standard star schema is a *fact table* that contains measure data. Radiating outward from the fact table like the points of a star are multiple dimension tables. *Dimension tables* contain attribute data, such as the names of customers and territories. The fact table is connected, or joined, to each of the dimension tables, but the dimension tables are connected only to the fact table. This schema differs from that of many conventional relational databases where many tables are inter-joined.

An advantage of the Epiphany star schema is that it allows an organization's EpiCenter to be regenerated when its business model changes without the need to replace the data warehouse. The Epiphany star schema also facilitates the creation of new EpiCenters for an organization (data from existing EpiCenters can be imported).

The Epiphany star schema subsumes the standard star schema into a larger hierarchical organization (known as a *constellation*) that allows for the sharing of dimension tables by a set of similar fact tables. A constellation is a grouping mechanism for like-structured fact tables.
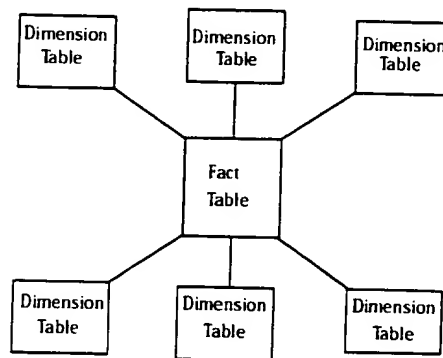
Figure 1-1 Standard Star Schema

As shown in the block diagram in Figure 1-2, page 4, the EpiCenter is the top-level organizing principle. Each Epiphany site has at least one EpiCenter. An EpiCenter is organized into one or more constellations, and a constellation consists of a set of similar facts. Dimension tables are shared by multiple constellations within the EpiCenter.

The EpiCenter

An EpiCenter is comprised of EpiMeta and EpiMart. EpiMeta refers to all of the Epiphany system's metadata tables. (*Metadata* is information about data, not data itself.) EpiMeta defines the schema for the EpiMart tables that will contain the actual extracted, organized data, such as customer, product, and order data. An EpiCenter is an EpiMeta database with its associated internal link to an EpiMart.

The EpiMart consists of fact, dimension, and staging tables. The fact and dimension tables contain actual customer data. Staging tables, which are discussed in Chapter 2, are the first entry point of raw data from the source systems into the EpiMart—an interim stop before it reaches the EpiMart's tables.
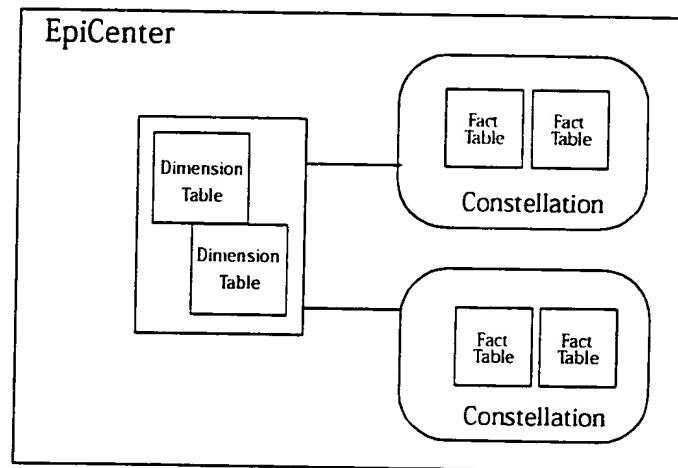
Figure 1-2 The Epiphany Star Schema

To facilitate the building of EpiCenters, Epiphany provides the EpiCenter Enterprise Manager, also called EpiCenter Manager. This is a Microsoft Windows application with an Explorer-like hierarchical structure (see Figure 1-3). The person who designs an EpiCenter (or EpiCenters) for a site uses the EpiCenter Manager's graphical user interface to define the schema for the EpiMart tables.
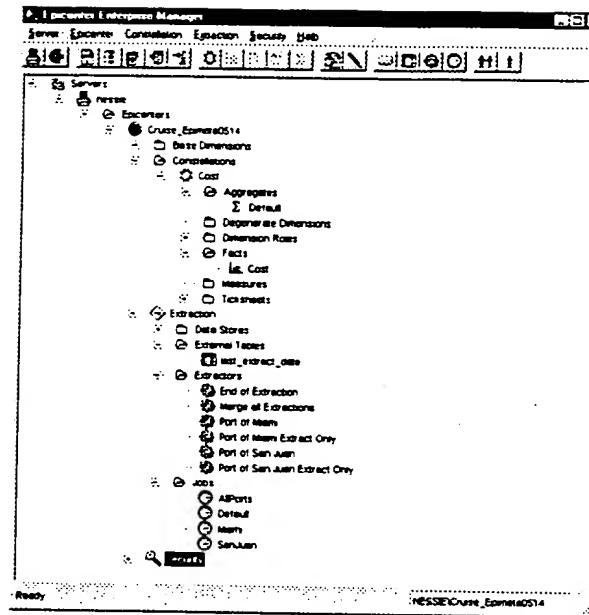
Figure 1-3 EpiCenter Enterprise Manager Window

Dimension Tables

A *base dimension table* consists of dimension columns that hold the actual attributes extracted from the source system. The Epiphany schema allows dimensions to be shared by all of the constellations within an EpiCenter. This is accomplished through the use of base dimension tables that serve as master tables for the entire EpiCenter, and dimension tables within constellations that are assigned "roles," or aliases. The latter are called dimension roles.

Dimension Roles

A row in a fact table may have several foreign keys that point to the same base dimension table, but the role usage differs. (A *foreign key* is a reference from one table to another table.) For example, a dimension role within an Order constellation and a dimension role within a Customer Support constellation

may correspond to the same underlying Master Product List table for data. Within the constellation, these dimension roles may be assigned the role of Product Order List or Customer Support Product List. Multiple roles within a single constellation can refer to the same base dimension.

Dimension roles are defined within a constellation, and all fact tables in that constellation inherit the dimension role. Dimension roles within constellations point to their associated base dimension table for their data. (A dimension role always has an associated base dimension table.)

Date and Transtype Dimensions

The Date dimension and Transtype (transaction type) dimension are assumed common to all constellations, and therefore are automatically provided as base dimension tables. To ensure consistent treatment of time, Epiphany uses a single date dimension throughout the system.

Transaction type dimensions are necessary because the EpiCenter must be able to distinguish among different rows in a single fact table, such as shipping transactions versus bookings, and bookings versus booked returns.

Attributes

Attributes describe a dimension. Some attributes, such as Customer Name, are free-text descriptions, but most have a discrete set of values. For example, a Territory dimension table may consist of its territory key and text that describes each of the organization's sales regions; such as, Eastern, Western, and Southern. It may also contain attributes for a region's subdivisions, such as City and State. Therefore, a hierarchical relationship exists in the dimension table in which the City attribute rolls up (aggregates) into State, which then rolls up into Region.

Region	State	City
Western	CA	Los Angeles
Western	WA	Seattle

Note: In the EpiMart, facts are quantifiable measurements, usually numbers. Attributes of dimensions are usually character strings.

Fact Tables

Fact tables are physical database tables that contain dimension role foreign keys and fact columns. A fact column is a single column in the fact table whose values are numeric, such as *net_price.* In the EpiCenter, a measure is an arithmetic combination of fact columns.

A row in a fact table represents a relationship among a set of values (each value is a separate field in the table). As shown in Figure 1-4, a row may exist for a specific customer number, product number, territory number, and date—all foreign keys that point to base dimension tables for their data—and fact columns, such as the number of units and the price per unit. The table may have millions of rows of data.

Customer	Product	Territory	Date	Marketing Program	No. of Units	Unit_ Price	Unit_ Cost

Fact Foreign Key Fields   Fact Column Fields
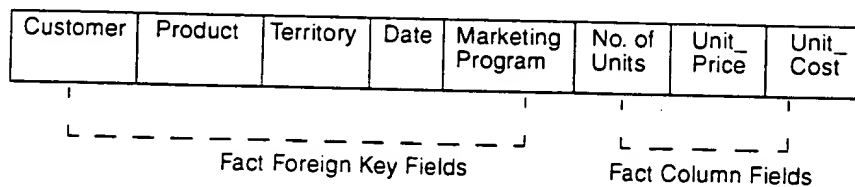
Figure 1-4 Sample Row in a Fact Table

For example, a fact table for Orders might have rows of data that contain fact columns and foreign keys that reference base dimensions, such as Business Unit and Territory. There may also be foreign keys for Customer_Bill-to and Customer_Ship-to that point to the Customer base dimension table. The dimension role metadata defines these foreign keys. (A fact table may have as many foreign keys to a single base dimension table as the business model requires.)

Each of the fact columns of a fact table can be totaled for any dimension and presented in report and chart format.

The Epiphany database organization allows users to make flexible queries. For example, a Clarity user can query the EpiMart for the names of customers who attended a Boston '96 trade show to determine the dollar amount of orders these attendees purchased as a result of this show.

Epiphany System Overview

An Epiphany site's source data may reside in any number of database management source systems, both relational and non-relational. Examples of supported relational database management source systems (RDBMS) and their related software are—

- telemarketing applications (ACT! and Aurum)

- sales force automation (Siebel and Onyx)

- enterprise resource planning (Oracle Financials, PeopleSoft, SAP R/3)

- customer support applications (Clarify and Vantive)

As shown in Figure 1-5, at the back end of the Epiphany system, data flows from the source databases into the RDBMS server database where the Epiphany database, called the EpiMart, resides. The Epiphany system accesses the source data in a read-only fashion; no changes to an existing source system are necessary. Database updating occurs as part of the database extraction process, which is usually run on a nightly basis.

At the front end, users open a ticksheet in a Web browser on any computer system that supports a JavaScript-enabled browser and select options for the kind of data they want to display. (A ticksheet is user-interface form in Epiphany that allows end users to construct queries; it is called a *ticksheet* because users select, or tick, items on a page.)The Web browser communicates with the Web server, which runs a Java program that queries the RDBMS server for the relevant information. The report is quickly generated and presented as an HTML document. The Epiphany Application Server is the Windows NT Service that connects with a Web server and delivers Epiphany ticksheets and reports. In the Epiphany system overview, it logically sits between the RDBMS and the Web server.

FRONT END

| Web Browser |
| (Ticksheet) |

HTTP

Web
Server

Java

Epiphany
Application
Server

- -

BACK END

RDBMS
Server

Epiphany database
(EpiMart)

ODBC Layer

Source System   Source System   Source System
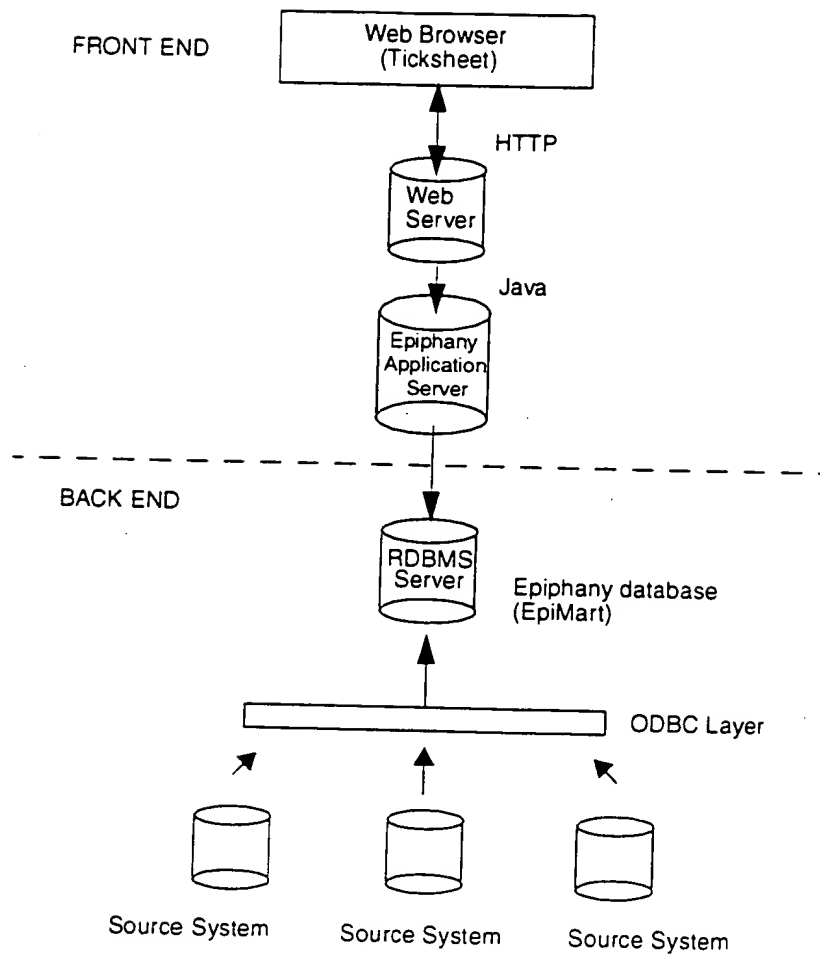
Figure 1-5 Epiphany System Overview

Although Epiphany's back-end tools can be used to generate and populate an industry-standard star schema, the power of the system derives from its integration with Epiphany's end-user Enterprise Relationship Management (ERM) applications. In order to achieve a robust, consistent application suite, each component interacts with a common metadata repository called EpiMeta.

EpiMeta is built on top of a relational database engine. EpiMeta's tables store all persistent aspects of an Epiphany implementation. When you configure an Epiphany system (using EpiCenter Manager), these are the tables you write to. Each of Epiphany's ERM applications is a consumer of this metadata, which determines the appearance of the applications, as well as other aspects of their behavior.

CHAPTER TWO

Epiphany Database Extraction

The goal of the extraction process is to extract raw data from data source systems and to place it into EpiMart tables where the data is accessible for query by front-end users. The extraction process, which is usually performed on a nightly basis, involves the use of these components:

- **EpiCenter Manager.** In addition to using EpiCenter Manager to define your site's star schema and constellations, you will use it to define extraction jobs. See Chapter 3, EpiCenter Manager, for Instructions.

- **EpiCenters** (EpiMeta and EpiMart). EpiMeta is the metadata database that defines the star schema and the semantics applicable to an organization. (*Semantics* refers to the means by which data is interpreted for an organization in terms its business processes.) EpiMeta also contains data related to the extraction jobs.

  EpiMart consists of fact, dimension, and staging tables. The fact and base dimension tables hold the data of the star schema (actual customer data), and the staging tables help to populate them.

- **Semantics** (types/templates and instances). At a high level, a semantic type reflects the semantics of a company's business processes. During data extraction, semantic instances, which are

post-compilation SQL programs, are used to merge data with the proper semantics into data loaded during previous extractions. See *Data Merging* on page 16 and *Semantic Instance Extraction Step* on page 25 for definitions of semantic-related terms.

- **EpiChannel.** This is the Epiphany extraction program that executes jobs and logs job activity. See *Running Jobs: EpiChannel* on page 37 and *EpiChannel Debugging* on page 44 for more information.

- **Database administration tools.** These tools are supplied by the database vendor to administer the physical characteristics of the databases and the interconnections among them. (Please refer to your database vendor's documentation for information about these tools.)

The major topics covered in this chapter are—

- Extraction Phases: An Overview (*see page 15*)

- Data Stores (*see page 16*)

- The Role of Jobs in Extraction (*see page 18*)

- Extractors (*see page 20*)

- System Calls (*see page 26*)

- Aggregate Building (*see page 26*)

- Custom Fact Indexing (see page 33)

- The UNKNOWN Dimension Row (*see page 34*)

- Running Jobs: EpiChannel (*see page 37*)

- EpiChannel Debugging (*see page 44*)

- EpiChannel Output (*see page 52*)

- Monitoring Jobs (*see page 53*)

- Mirroring: A and B Tables (*see page 55*)

The first topic addressed is one of the Epiphany system's basic concepts.

Basic Concept: The Epiphany System Is Metadata

All of the control information for an EpiCenter is stored in a single metadata repository called EpiMeta. EpiMeta represents a transactional, fully relational model of over one hundred and fifty tables that have complicated declarative referential integrity constraints. Epiphany provides tools such as Web Builder and EpiCenter Manager for configuring this metadata without the need to write to, or even know about, the underlying data structures.

The EpiMeta tables of metadata control all aspects of the system including:

- Star schema structure

- Extraction workflow and semantics

- Aggregation and other performance enhancements

- Business calculations (or measures)

- User interface layout

- Security information

- Saved Report objects

EpiMart contains the customer data at an implementation. The schema of tables in EpiMart is completely determined by the schema metadata in EpiMeta. The contents of EpiMart tables are determined by the instructions contained in the extraction metadata tables. Theoretically, the contents of EpiMeta alone can be used to reconstruct an equivalent EpiCenter; thus re-extraction from the source system should result in an identical EpiMart. Because EpiMeta effectively defines an EpiCenter, Epiphany provides an export/import utility to back up metadata, or to transfer subsets of metadata between EpiCenters. (see Appendix H, *Export/Import of Metadata*, for more information.)

Each Epiphany tool reads from and/or writes to EpiMeta. The interaction between tools and EpiMeta is described below:

EpiCenter Manager   Writes schema metadata, extraction metadata, security metadata, and aggregate definition metadata.

EpiChannel   Reads schema, extraction, semantic metadata. Writes logging data about the extraction.

Aggbuilder   Aggregation, the pre-calculation of selected facts to speed up the front-end query process, is performed by the Aggbuilder program. Aggbuilder reads schema metadata and aggregate definition metadata, and writes aggregate navigation metadata.

     Aggregate navigation is the process by which the Epiphany query machinery determines the optimal aggregate table to use to satisfy an application's request for information. (The *query machinery* is the component of the Epiphany Application Server that communicates with EpiMart and actually issues SQL statements against the DBMS.)

Web Builder   Web Builder is used for configuring user-interface metadata, including measures, ticksheets, and dictionary entries. It reads schema metadata and writes measure and ticksheet metadata.

Epiphany Application Suite

     Clarity and Relevance applications read schema, measure, ticksheet, and aggregate navigation metadata. They write saved object metadata.

Each table in EpiMeta defines an integer primary key. The database engine provides the actual unique primary key values for each row in the metadata. All foreign keys between metadata tables are accomplished with these integer

columns. The benefit of using non-natural primary keys is that all other columns in the metadata can be changed without affecting relationships in the model. Note that each foreign key uses declarative referential integrity to ensure consistent metadata. Additionally, EpiMeta enforces other declarative integrity constraints based on the correct interpretation of these tables.

Extraction Phases: An Overview

There are three main phases of the extraction process—load (pull/push), data merging, and aggregate building. Each phase is related to the metadata that defines the tables and their columns.

Load Phase

During the load, or pull/push extraction phase, data is extracted from a source database or file in raw format and loaded into interim tables (staging tables or external tables). A *staging table* is a table that contains all of the fields required by a single base dimension or fact table. Staging tables hold the data in preparation for the second phase of extraction: data merging. An *external table* is a table built in EpiMart (and thus internal to the Epiphany system), which usually serves as a temporary table during extraction. The external table, which is similar to a staging table, receives the data directly.

Staging and external tables have a schema similar to the target EpiMart tables (Epiphany's Adaptive Schema Generator generates both simultaneously). The *Adaptive Schema Generator* is a component of EpiCenter Manager that builds the tables in EpiMart using the schema metadata definitions in EpiMeta.

When you run the EpiChannel program to start the extraction process, it reads the metadata for the job you selected and begins executing extractor steps. An *extractor* logically specifies a series of steps that move data from the input to the output data source. Occasionally, an extractor step is directed to populate external tables rather than staging tables. External tables serve as intermediary tables when more complicated multi-staged extraction is required.

Data Merging

The second main phase of extraction is data merging. After the staging tables are fully loaded, semantic instances merge the new data in the staging tables into the EpiMart database tables. A *semantic instance* is the usage of a generic semantic type on one fact or dimension table during part of an extraction job. As part of defining tables, you assign each an Epiphany-defined semantic type. A *semantic type* refers to the logical business process for which a generic SQL program called a *semantic template* will be applied.

The Epiphany extraction program, EpiChannel, executes the semantic instance at the appropriate time during an extraction job.

Aggregate Building

The third and final extraction phase is aggregate building. Aggregates are pre-calculated and pre-stored summaries of data that significantly accelerate the front-end *query machinery*. The query machinery is the component of the Epiphany Application Server that communicates with EpiMart and actually issues SQL statements against the RDBMS.

You will use EpiCenter Manager to define which facts are aggregated for groups of dimensions. The Aggregate Builder program, which typically runs immediately following the data extraction, performs roll-ups, or aggregations, on the groups and includes them as aggregates in the EpiMart tables.

At the conclusion of the extraction phase, the extracted data resides in the proper tables and columns in the EpiMart. The data has been extracted from the source system or file data source (called a data store) and placed in the data store on the RDBMS server.

Data Stores

A *data store* is a logical location of data that functions as either as a source or sink within an EpiCenter. Data stores include relational database connections, as well as files and directories. The concept of a data store is integral to the Epiphany extraction process. A data store represents the physical location of a source of data that the Epiphany system can read from and/or write to.

Typically, the input data store is a source system database or file, and the output data store is EpiMart, which holds customer data, on the RDBMS server. As part of the installation procedure, you will create a new, empty database for EpiMart (and EpiMeta).

As shown in Figure 2-1, the Epiphany system extracts the raw data from a site's source system—for example, a RDBMS server, a generic ODBC source system, or a flat file—via Open Database Connectivity (ODBC). ODBC is an abstraction layer that provides a common interface to many databases. This data is read by various ODBC drivers and written using the target database system's API. The Epiphany system can access data if an ODBC driver exists.

EpiChannel, the Epiphany extraction program that executes jobs and logs job activity, opens an ODBC connection to the source system and a database library connection to the target server and initializes tables.
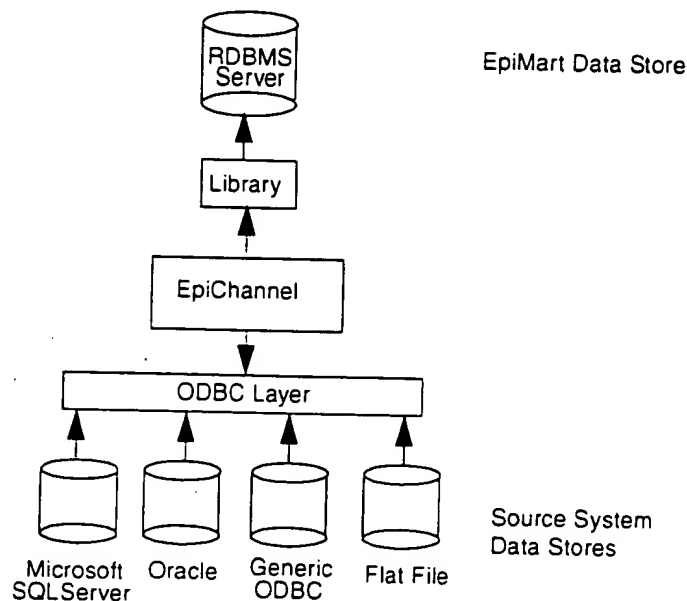
Figure 2-1 The Back End of the Epiphany System

You will use EpiCenter Manager to configure the data stores as part of setting up your site's extraction process. When an EpiCenter is created, it has three default data stores:

- *EpiMart*

  A special data store for the actual database on which Epiphany's front-end applications will run.

- *JobFileLog*

  Specifies the data store for EpiChannel job logs.

- *LoggingDB*

  Specifies the data store for the EpiChannel logs.

You may, however, set up as many data stores in EpiCenter Manager as necessary. For example, operational systems for different divisions within an EpiCenter may have the same fact and dimension tables (EpiMart), but draw data from different source systems. In this case, you would create a data store for each input source system. For more information, see *Data Stores* on page 90.

> *Note:* The EpiMart is the data store, or database, that you set up for your data warehouse. The EpiMeta holds the description of your data (the extraction jobs, the ticksheets, and the star schema definitions). The EpiCenter is a combination of EpiMart and EpiMeta.

The Role of Jobs in Extraction

The Epiphany extraction process works through the running of jobs. A site defines the jobs needed to extract its data and runs these jobs on a regular basis to update the EpiMart. A job is basically a batch of work to be performed as a unit. The work consists of an ordered list of job steps, each of which is either an extractor or a system call. System call job steps may be interspersed with extractor job steps in any order. (See Figure 2-2, page 19 and Figure 2-3, page 21.)

A simple extraction may consist of one job, which has multiple steps. Each step utilizes the data produced by the previous step. A multi-stage extraction, however, may consist of multiple jobs. The EpiCenter Manager provides a graphical user interface (GUI) for defining job steps and the job input and output data stores. The actual job steps (extractors and system calls) need to be customized for each site.

After the jobs have been defined and the EpiCenter schema generated, the database administrator can invoke the EpiChannel (**extract.exe**) command to begin the extraction process that will populate a new EpiMart, or update an existing one.

Job 1

Step 1
Step 2
Step n

Job 2

Step 1
Step 2
Step n

Job n

Step 1
Step 2
Step n
End of Extraction Extractor

Figure 2-2 Job Work Flow

Extractors

An extractor is one or more sets of SQL operations that will be executed against a particular source and destination data store. These same sets or groups of SQL operations may be shared by another extractor that applies them to different data stores. The SQL operations may be either SQL statements or semantic instances. In general, SQL statements are used to extract data, and semantic instances are used to merge data with the proper semantics into data that has been loaded during previous extractions.

Semantic instances are designed to work on an idealized star schema. Macros in these semantic instances are translated at extraction time to the tables and columns in the target star schema. (The fact or dimension table listed in the Semantic Instance dialog box (Figure 3-20, page 101) in EpiCenter Manager determines which column names are actually used in the final SQL.) This enables the complex transformation logic in a semantic instance to be applied to star schemas other than the one for which the semantic instance was first used.

As shown in the job work flow (see Figure 2-3), a job step may be either a system call or an extractor. An *extractor* is a list of extractor steps (and input and output data stores) that move data from the input to the output data store. An *extractor step* is a single step of an extractor, which always points to an extraction group. An *extraction group* is a set of extraction steps. An *extraction step* is a single atomic extraction operation that can be either a SQL statement or a semantic instance.

Extraction steps and groups are both modular. After you define them, you may reuse them in other jobs. (The Extractor Steps dialog box (Figure 3-18, page 97) in EpiCenter Manager is where you define a new, or select an existing extractor for a job.)

> *Note:* An extractor has associated input and output data stores and exists independently of any extraction step.
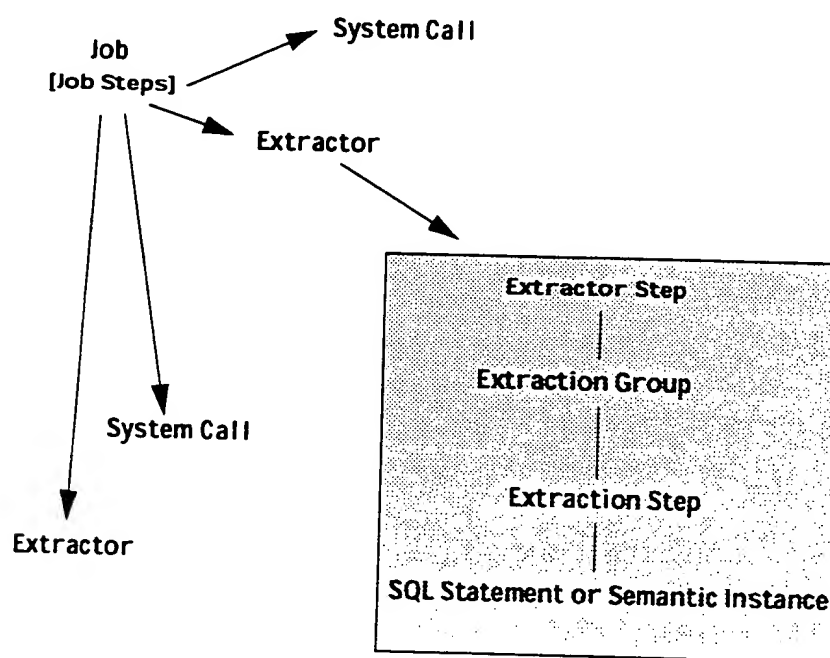
Figure 2-3 Job Structure

An extractor can move data from a single source data store to a single destination data store. A job may consist of multiple extractors, however, each with its own independent input and output data stores.

Jobs may also share extractors, applying the same logic in different orders—if the jobs' data stores have common structures. For example, a job that runs

Monday through Friday could share the same extractor with a Weekend version of the same job in which the only difference is that the Weekend job has an extra step, an extractor that updates rarely changing tables.

The two kinds of extraction steps that may comprise an extractor step are discussed below: SQL statement and semantic instance.

SQL Statement Extraction Step

An SQL statement extraction step may be either a stand-alone SQL statement or a pull/push statement.

A stand-alone SQL statement is executed against either the source or destination data store for that extractor and is usually specific to the data store's type. These SQL statements are usually issued to achieve a side effect, such as dumping the transaction log or setting up environment variables. Any returned results are discarded.

A pull/push SQL statement is a type of extraction step that issues SQL against an input data store (and is interpreted according to the rules of that database's engine). The returned results are inserted (pushed) into the destination table for that step, either a staging table or an external table (the output data store of the extractor).

The result set columns are mapped either to columns in the destination tables or to other functions. The mapping is a case-insensitive exact match by name of the available columns, or function results, to the destination table columns.[2]

At this stage of the extraction, each destination table column must have a matched value. If not, the mapping fails, the SQL statement results are discarded, and the statement fails. Extra columns that are available from the SQL statement or functions, but which are not used in the destination system, are not considered to be errors. (SQL statements with extra columns will have their mapping displayed in the EpiChannel log if you set the verbosity level for EpiChannel high enough; see *EpiChannel Debugging Levels* on page 50.)

---

[2] Special characters such as the *at* symbol (@) are removed when this match is performed.

Once mapped, the rows from extraction SQL statements are fetched into EpiChannel memory where they are verified for field width and forwarded to any function that consumes the field. The mapped results are forwarded to the destination database.

All operations on the source data store or the EpiMeta data store occur through the appropriate native library for the EpiCenter. The inserts are batched so that the extraction statements maintain a count of rows fetched, rows forwarded, and rows committed. The batch is fully sent and committed before the extraction SQL statement ends and the next job step may begin.

Any error, as determined by the database engine evaluating the SQL, results in the SQL statement being considered in error. If the action plan for the SQL statement (as selected in the General tab of the SQL statement dialog box) is to abort, then the job halts without executing any other SQL operations or job steps.

SQL Macros

A SQL statement may contain extraction macros that are expanded before the SQL is executed. Some of these macros provide database-independent functionality by expanding in different ways for different databases, and some use the metadata to cause the SQL to reflect the edits made in EpiCenter Manager. These macros are described in Appendix B.

Staging Tables

The most common destination for an extraction statement is a staging table. As mentioned, a staging table is a metadata table that contains all of the fields required by a single base dimension or fact table. Unlike the real fact and base dimension tables in the EpiMart, the staging tables have dimension columns that contain key fields from the customer's system, rather than Epiphany-generated foreign keys. Staging tables typically contain data extracted by only one job, rather than accumulating data over time.

Once a staging table is populated, semantic instances are used to merge the new data in the staging table into the existing data. A semantic instance is a series of statements, similar to a program. These statements maintain the integrity and additive nature of the columns in the facts tables and maintain correct references to dimensions even though the dimension may change from one reporting of a fact to another.

See Appendix F, *Writing Staging SQL Statements*, for more information.

External Tables

Extraction statements are sometimes directed to load external tables rather than staging tables. External tables serve as intermediary tables when more complicated multi-staged extraction is required. such as for the grouping of data, dividing values in a field into bins, or joining with data from another source system. They may also serve as holding fields used by non-Epiphany data transformation tools, such as third-party name and address cleansing tools. In the case of cleansing tools. extractors merge the cleansed names and addresses with other tables extracted from the source databases into the EpiMart tables.

EpiCenter Manager provides the extractor named *End of Extraction* for each EpiCenter by default. This extractor consists of two steps:

1. Issue SQL against the EpiMart to determine the date of the last extraction to be displayed to users.

2. Toggle the current EpiMart from A to B, or vice versa.

Step 1 usually consists of SQL statements that load the external table named *last_extract_date*. The *last_extract_date* table is provided by Epiphany, and is recommended for use at your site.

Step 2 updates the *config_master* value *last_extract_date* with the latest value received from Step 1. It then updates the *config_master* value called *current_datamart*, switching it between A and B.

> *Note:* The last extraction date setting is the date that appears as the *Data is valid as of...* in Clarity reports. It is determined by an option in the EpiCenter Manager's Configuration dialog box (described in Appendix C).

Switching the focus between the A and B tables involves an update to a single row in the database (no tables or databases are actually moved). See *Mirroring: A and B Tables* on page 55 for a discussion of A and B tables.

See *Using External Tables as Inputs to Staging Queries* on page 266 for more information about external tables.

Semantic Instance Extraction Step

As mentioned, the Epiphany extraction process (exclusive of aggregation) is a two-phase operation:

- load (pull/push) phase

  The load phase refers to the loading of staging tables, which are the first entry points of data into EpiMart.

- data merging phase

  The data merging phase refers to the loading of staging tables into the EpiMart's permanent base tables.

This process of moving data from staging tables into the EpiMart's base tables is called *semantic transformation.* Proper configuration of the data merging phase of extraction, requires an understanding of the underlying meaning, or semantics, of the data.

Epiphany provides a set of semantic types/semantic templates that you may use without having to be concerned with their internal code. A *semantic template* is a generic SQL program intended to accomplish specific business rules during extraction (these rules are referred to as *semantic types*). A semantic template program contains SQL with generic table names; it does not refer to actual column or table names. Only when the semantic template is applied to a base dimension table or fact table does the SQL contained within it refer to actual column and table names.

When a semantic template is combined with a base dimension table or a fact table, the result is a semantic instance. A *semantic instance* is a valid SQL program that can be run against EpiMart.

Applying a Semantic Type

Epiphany provides a set of semantic types for each EpiCenter. When you use the Semantic Instance dialog box in EpiCenter Manager to assign a semantic instance, you select one of the semantic type options. The list of available options is determined by whether the object that the semantic instance will operate upon is a base dimension table or a fact table. Your selection assigns the semantic type to the table (a row is added). The implementation, as opposed to the assignment, of a semantic type is stored in metadata tables and is invoked by EpiChannel at run time.

Dimension semantic types have two purposes: creating base dimension tables and creating dimension mapping tables. Dimension mapping tables are used to convert dimension source system keys (*sskey*) in fact tables to Epiphany dimension keys. Dimension semantic types are also responsible for proper indexing of these two tables.

Fact semantic types are normally used to populate and Index new fact base tables. For descriptions of the fact and dimension semantic types, see Appendix G. See Appendix F, *Writing Staging SQL Statements*, for a description of source system keys.

System Calls

While SQL-based transformations may be sufficient to extract from simple databases, sites with more complex databases may require multi-stages and additional commands, such as lookup tables, aggregation splits, gathering data into ranges (binning), and duplicate detection. For this purpose, you may use system calls, which are executed during a job as if invoked from the console's DOS command line.

In a multi-stage extraction, files might need to be moved, decompressed, de-encrypted, or purged. Although these steps could be placed before or after extraction statements, they may need to be placed in-between SQL-based steps. EpiChannel coordinates the execution of system calls between extractors.

A complex job might require the following steps:

1. The invocation of a system call to uncompress a file.

2. The invocation of an extractor to populate external tables.

3. A system call to invoke third-party software to cleanse the names in the external tables.

4. Extractors to merge the cleansed names with other tables extracted from the original and other databases into the EpiMart.

Many system calls need to reference the location of databases or files. In the above examples, Steps 1, 2, and 3 need to share the file and database names for the data they pass to each other. Although these names could be hard-coded, this is not the best approach; for example:

- The system call could break if the database login information changed, and the system call text was not modified accordingly.

- The system call cannot be easily reused.

- Some file names need to be adjusted from run to run so that useful intermediate data is not overwritten.

Job data store roles provide an alternative to hard coding database and file information in system calls. Roles are referenced by macros in systems calls, and these macros look to the job to see what data stores are associated with the roles. The information from the matching data store is replaced by the macro. For example, the system call

```
mkdir $$DIRNAME[Working Dir]/aggbuilder_temp
```

would have the $$DIRNAME macro expanded to the MS-DOS file path that EpiChannel is using as a *working directory*. The **mkdir** command then creates a subdirectory in this location. Because EpiChannel generates a unique directory name for itself for each run, the above command would create a corresponding unique subdirectory for use by the Aggregate Builder program (Aggbuilder), and this directory would be moved/purged along with EpiChannel logs.

See Appendix B for more information about Epiphany macros.

Note: All system calls are expected to have an exit code of zero, but you can use the Add Job Step dialog box (see Figure 3-23, page 106) in EpiCenter Manager to set the *On Error* type to Abort or Ignore, as appropriate.

Aggregate Building

Creating aggregates speeds up system performance at the front-end of the Epiphany system. This is because some queries simply require summary data, which can be pre-computed by the system rather than being recomputed for each query. For example, assume that a site has 5,000 products categorized by ten product types. Clarity users may request reports that show data aggregated by product type. If there is a grouping by product type, the Epiphany aggregation program creates a table with ten rows (one for each product type) and pre-calculates "rolled-up" (combined) fact aggregates based on this smaller dimension table. By using significantly smaller fact aggregates, the system can generate reports considerably faster.

When EpiMart is successfully populated, the rows in the fact tables contain foreign keys to dimension table rows. In Epiphany, all dimension primary keys are stored as integers. For instance, fact rows may resemble the following in the fact table Order_O:

Order_O

Customer_key	Product_key	Salesperson_key	Total Sale
6	8	5	$20
5	6	6	$30

whereas the dimension tables may look like these:

Customer_O

Customer_key	Customer Name	Region
5	ABC Company	West
6	Bill's Lighting	West
7	Fred's General Store	East

Product_O

Product_key	Product Name	Platform
6	Product A1	Windows
7	SuperX	Mac
8	Smash-Em	Windows

Salesperson_O

Salesperson _key	Salesperson Name
5	Gene
6	Sue

All fact and dimension tables that end with _O (underscore zero) are called base tables in EpiMart. They contain raw data at the level that is extracted from the source system.

A base dimension is a physical dimension table in EpiMart, such as Customer or Product. Base dimension tables contain the actual dimensional values that are available for reporting or filtering. Attributes, such as Customer Region or Product Platform, can be rolled up. For example, end users often want to create reports of total sales by region. Suppose that Order_O has hundreds of

thousands of records, and Customer_O has many thousands of records, and that there are only three different regions in the Region field. A report of sales by region against Order_O and Customer_O will need to aggregate over thousands of records, causing the report to return answers very slowly. However, a pre-built aggregate with three (rolled-up) rows that already has the total sales by regions could answer the same question very quickly.

After configuring an EpiCenter, the following aggregates might be built:

Order_17

Customer_key	Total Sale
1	$50

Customer_1

Customer_key	Region
1	West
2	East

Note the following:

- The *Customer_1* table contains only the rolled-up field(s). The number of rows has been reduced so that each row contains a distinct Region. The table name is the same as the base table, with a different number at the end.

- The *Order_17* table contains the same total sales dollar amount, but the number of rows has been reduced (in a larger example, this reduction could be extremely significant). The name of the table is the same as the base table, with a different number at the end.

- The foreign key *Customer_key* has the same name as in the base tables above, but the numbers differ. It is important to recognize that the *Customer_key* field in *Order_17* can only be joined with the *Customer_1* table, since *Order_17* is at the granularity of Regions, not Customers.

See *Defining Dimension Aggregates* on page 71 for a description of how tables such as *Customer_1* are built using EpiCenter Manager.

The Aggbuilder Program

Aggregate building is performed by Aggbuilder (**agg.exe**), a program similar to EpiChannel. Aggbuilder uses the same command line, EpiMeta database, and job definition table as EpiChannel. Aggbuilder, however, completely ignores the job steps (that is, system calls and extractors). It uses the job table only for an indication of the *log database* and *working directory*. The structure of Aggbuilder log files is similar to that of EpiChannel except that the *log* directory name is *agg_timestamp* instead of *chn_timestamp*.

The easiest way to invoke aggregate building is via a system call that uses some of the system-call macros to isolate details. To invoke Aggbuilder, add a system call similar to this after the extraction and semantic instances steps, but before any use of the extractor called *End of Extraction*:

```
$$AGG $$EXC_ARGS -j agg_job_name
```

where **agg_job_name** is the name of the job with the proper logging information.

To log to the same places as the current job, use a system call such as:

```
$$AGG $$EXC_ARGS -j $$JOBNAME
```

The Aggregate Building Process

When the Aggbuilder program is executed during an extraction job, the following actions are taken:

> *Note:* All actions occur in the set of mirrored tables (A or B) that is *not* currently active. See *Mirroring: A and B Tables* on page 55 for more information.

1. Dimension aggregates are built. Each column set for a base dimension will become a dimension aggregate as long as that column set is included in an aggregate group via EpiCenter Manager's Aggregate Group dialog box (see *Aggregation and Aggregate Grouping* on page 81). Indexes are built for these tables as appropriate.

2. For each fact table, all enabled aggregate groups to which that fact belongs are examined. Aggbuilder determines all possible combinations to be built based on the dimension role definitions and creates unique table names. Fact aggregates are physically built and indexed.

3. Aggbuilder records what it has built in a set of read-only metadata tables in EpiMeta. You may use EpiCenter Manager to browse these lists. The Epiphany Application Server uses these same lists at runtime to decide which aggregates to use to satisfy an end-user query.

4. All activity is logged to Epiphany's logging tables.

Custom Fact Indexing

EpiCenter supports two methods that dramatically improve query performance: indexing and aggregation. Each accelerates different classes of queries.

In general, the presence of aggregate tables accelerates queries that answer high-level business questions, such as Sales by Business Unit and Fiscal Year. Deep drill-down queries, or highly filtered queries that ask for a small subset of data, are typically enhanced by an index. A fact base table is usually the largest table in EpiMart. Even tables with millions of rows can be accessed quickly via an index when only a small number of data pages is needed to service a query.

A given Epiphany query uses only a *single* fact table (either base or aggregate). Typically, as a user drills down into a result set, the physical table that services each query moves from a high-level aggregate (with few rows) to a larger aggregate (possibly using an index on the aggregate). For the user's response time to be acceptable in each of these cases, you must properly configure both aggregates (as described in the previous section) and custom indexes.

All aggregates are indexed in the same manner as the base table (insofar as the aggregate has the same dimensionality as the index). For example, if the base table has an index on *customer_key, product_key,* and *territory_key,* then an aggregate that contains only the Customer and Territory dimensions will have an index on *customer_key* and *territory_key.*

> *Note:* If two different indexes on the base table result in the same index on the aggregate, then the aggregate index is built only once.

By default, each fact table has one index built on its dimensional keys during extraction (on Microsoft SQLServer this index is also the CLUSTERED index). For all fact tables, the leading term of the index is *date_key* because a wide class of queries filters on the time dimension. For example, if a system contains data for multiple years and a user asks for a single calendar month, the database almost certainly will use an index when it services this query (whether via the base table or an

aggregate). All other dimension roles in the fact table are also included in this index. The order in which they appear is determined by their order in the constellation within EpiCenter Manager's directory tree.

To re-arrange this order, right-click a dimension role folder, and select Up or Down from the pop-up menu.

A custom index is the metadata definition of indexes to build on fact tables in EpiMart. Normally, each fact table is given a single index. You can use EpiCenter Manager (the Custom Index tab in the Fact Table dialog box) to:

- configure all other indexes on the fact table.

- assign each index a logical name, although the name is not used when building the physical index.

- choose the dimension roles that make up the index, and their order.

    Note: You will use the Custom Fact Index semantic type to actually build the indexes. (See Appendix G, *Semantic Types*.)

It often makes sense to build several different custom indexes, each with a different dimension role as the leading term. If, for instance, end users often filter on attributes of the Customer and Product dimensions, then you could build a custom index on the combination (*customer_key, product_key*) and another on *product_key* by itself. There is no need for the second index to also include *customer_key* because the first index could easily service joint customer/product queries.

The UNKNOWN Dimension Row

When a fact staging table is loaded during an extraction, its dimension source system key columns (columns whose names end with *sskey*) normally refer to rows in the base dimension staging tables. (The column name being referred to in the base dimension staging table also ends with *sskey*.) These source system keys are the actual values used to form the relationships on the source system between the fact and dimension source tables.

Some source systems do not enforce referential integrity, however, which may result in source system fact tables that have "dangling references," or references to non-existent dimension values. Another possible source of a dangling reference is an optional or null foreign key in the source system. In EpiMart, all fact dimension keys must point to valid base dimension entries; and there should be no dangling references.

To prevent dangling references, each base dimension in EpiMart is populated automatically with a single special row called the UNKNOWN row. This row is always available for mapping fact dimension keys that would otherwise not point to a valid base dimension row.

When writing fact staging SQL statements, you can refer explicitly to this special row by populating the *sskey* dimension role column with the special string UNKNOWN. For instance, on a Microsoft SQLServer source system, you might use the ISNULL() function as follows to translate all null values to UNKNOWN:

```
SELECT

    ...

    ISNULL(cust_code, 'UNKNOWN') customer_sskey

    ...

FROM

    ...
```

Referring to the UNKNOWN Row during Extraction

For a base dimension such as Customer that has dimension columns *customer_name* and *region*, the UNKNOWN row has actual values similar to any other row. By default, each dimension column assumes the literal value UNKNOWN for each of its textual attributes. However, you may use EpiCenter Manager to override this default value so that end-user queries return a different value. To do so, enter a literal string (without quotes) in the Dimension Column dialog box (accessible from the General tab of the Base Dimension dialog box shown in Figure 3-6, page 70). Your string will be used instead of UNKNOWN.

The Update Unjoined Semantic Type

While extracting a fact staging table, one often does not know whether the values that appear in the dimension foreign key columns actually refer to dimension rows. While the technique described above for converting null values to the UNKNOWN string works for missing dimension foreign key values, it will not work when a dangling value appears. If no action is taken, then these fact rows will be lost during extraction since the first step of most fact semantic types involves an inner join between the fact staging table and the dimension mapping tables (see Appendix G, *Semantic Types*).

The Update Unjoined fact semantic type is specifically geared towards converting these dangling fact references to the special UNKNOWN value. Scheduling this semantic type (after the fact staging table has been loaded) during an extraction will cause a scan of the fact staging table with joins to each of the dimension mapping tables. All *sskey's* in the fact staging table that do not map to extracted dimension rows will be converted to UNKNOWN.

> *Note:* Execute the Update Unjoined fact semantic type before executing any other semantic types on this fact staging table.

In general, it makes sense to execute semantic types for base dimensions before any of the semantic types for facts (including Update Unjoined) because the base dimension semantic types produce the mapping tables needed by fact semantic types (see *Normal Extraction Order* below).

Normal Extraction Order

Epiphany's recommended sequence of events during an extraction is as follows:

1. Load fact staging tables.

2. Load base dimension staging tables.

3. Execute base dimension semantic Instances.

4. Execute fact semantic instances.

The reason for loading the fact staging tables before base dimension tables is subtle. If Epiphany is extracting from a live source system, then new dimension and fact rows could be created while the extraction occurs. If dimensions were extracted first, then when facts are extracted, a new dimension row (such as a new Customer) might have been created, but was missed. If a fact that refers to that new Customer is then extracted, the fact row shows up in the Epiphany system with an UNKNOWN Customer value (because that Customer row was missed by the extraction). By extracting the facts first, the system might not receive the fact row in this extraction, but will retrieve it in the next extraction.

Running Jobs: EpiChannel

EpiChannel is the Epiphany extraction program (**extract**) that you will run to populate your initial EpiMart and to update it on a regular basis.

EpiChannel first opens an ODBC connection in the source system and then opens a database library connection to the target system and initializes tables. It establishes start and stop date time boundaries and replaces certain special strings. After replacing these strings, EpiChannel executes a user-defined sequence of SQL statements or system calls.

The SQL is pre-processed to match warehouse and database vendor syntax. System calls are pre-processed to supply directory locations and database login information.

Before attempting the first job step, EpiChannel verifies the availability of databases and tables (if these options are selected in EpiCenter Manager's Job dialog box) and expands most macros. If these fail, the job does not run.

The job will halt when it encounters the first error in any step, unless that step has an error code set to print or ignore the error instead of to abort it. Error codes are set through EpiCenter Manager. You will use the SQL Statement dialog box (Figure 3-19, page 98) for extractors, and the Job dialog box (Figure 3-21, page 102) for system calls.

Whenever a job succeeds or fails, e-mail notification is automatically sent to the database administrator (or any designated list of e-mail recipients).

The EpiChannel Command Line

This section describes the command-line syntax you will use to invoke EpiChannel at the console's DOS prompt. These command-line parameters indicate the database with the job information (that you configured using EpiCenter Manager) and parameters related to interactive debugging. Command-line parameters may also specify the initial debugging level, the maximum selection limit, or that the jobs are to be trial runs.

To invoke EpiChannel, enter the **extract** command on the console's DOS command line followed by the parameters described below.

> *Note:* EpiChannel inherits the case sensitivity of the database engine.

```
extract
```

```
-h
-?
-v integer_verbosity_level
-I instance_name
-S server_name
-D db_name
-B db_vendor_name
-U username
-P password
-t
# row_count
-j job_name
```

where:

-h and **-?**	Display detailed on-line help.

-v *integer_verbosity_level*

> Runs the EpiChannel job in verbose mode, which displays the SQL on the screen. See *EpiChannel Debugging Levels* on page 50 for more information.

-I *instance_name*	Specifies which Registry instance key to use. This parameter is mutually exclusive with many of the other parameters because it causes them to be read from the Registry.
-S *server_name*	Specifies the name of the server where the EpiMart resides.
-D *db_name*	Specifies the name of the EpiMeta database.
-B *db_vendor name*	Specifies the name of the EpiMart database vendor.
-U *username*	Specifies the username for the EpiMeta database.
-P *password*	Specifies the password for the user of the EpiMeta database.
-t	For testing purposes, this is the trial-run option that simulates execution of the SQL on the source and destination databases.
# *row_count*	Specifies the maximum number of rows to fetch on any SQL statement.
-j *job_name*	Specifies the name of the Job as defined in the EpiMeta.

EpiChannel Registry Keys

The EpiChannel program uses the Registry keys shown in Table 2-1, which are located in the following Registry path on the local machine:
Software\Epiphany\Instances\instance_name.

In the table below *Default Instance* is the string value for Default in the Instances Registry key under the Epiphany Registry key.

Table 2-1 EpiChannel Registry Keys

Key	Value
1	Default Instance
2	[Default Instance]/DBVendor
3	[Default Instance]/Database
5	[Default Instance]/Server
6	[Default Instance]/Username
7	[Default Instance]/Password

By default, EpiChannel reads the Default Instance key and uses its value to open the Registry tree that holds all initialization parameters. You can override this by specifying a different name using the -I (*instance_name*) option.

For example:

```
extract -I name
```

forces EpiChannel to read all Registry values from
HKEY_LOCAL_MACHINE\... \Epiphany\Instances\instance_name.

Other EpiChannel command examples:

- A command (in a system call) in which all Registry variables are set:

  `extract -j job1`

- A command example in which no Registry variables are set:

  `$$EXC_CMD`

Output Files

Jobs may log to files or databases, and these logs can be directed to multiple destinations simultaneously: For example, one log could be placed on the local machine and another on a network server.

The execution of jobs is logged as a unit. Statistics are collected for job steps and for entire jobs. The location of the log is established by associating data stores with the *log* or *working directory* roles for a job, for which you will use the Job dialog box in EpiCenter Manager (see Figure 3-21, page 102).

Extracting New Rows Only

An efficient extraction process pulls only data from the source system that has changed or been added since the previous extraction. The way that the extraction process recognizes this subset of data depends on how the source system identifies changes in data. Epiphany supports the following methods of identifying changed data:

- Date or Date/Time fields in source rows

  A source row, or a table it joins to, contains a date or date/time indication of when it was inserted or last updated.

- Timestamp fields in source rows (Microsoft SQLServer only)

  Values can be "stamped" in some way to indicate their status. Epiphany provides Microsoft SQLServer timestamp fields for source rows. A source row, or a table it joins to, has a column that contains Microsoft SQLServer data type *timestamp*. The database engine automatically updates this column whenever the row is inserted or updated. This field does not

actually contain any record of the time that the update occurred, but rather values that always increment, such that any row edited after another row will have a higher timestamp.

- Highest numeric or string value in named columns in source rows

A particular column in a particular table is presumed to contain a value that is always increasing from row to row; for example, Order Number or Policy ID. For this method to be useful as a subset selection criterion, however, either the rows must never be updated, or else the column must be changed by the source system whenever an update occurs.

How EpiChannel Identifies Data To Be Extracted

The following topics describe how EpiChannel identifies which subset of data to extract:

- How EpiChannel reads values

EpiChannel attempts to get a consistent set of dates/timestamps/column values so that the same single moment in time is captured by the WHERE clauses generated by the extraction set identification macros. The extraction program minimizes the chance for changes to occur to one set of values, and not to another one, by reading all sets one after another with no intermediate actions. While this gives highly consistent values, it is possible that a change may occur between the time EpiChannel reads the first table/column and when it reads the last one, especially if multiple databases are involved.

The values read are stored as the "last" values only if the job commits its work. If any system call or SQL statement has an error that aborts the job, it is as if the value stamps had never been read or modified. In the unlikely event that a job aborts during the commit phase, the timestamps might not be synchronized with each other.

The date and timestamp values are read from a table-independent current value such as SYSDATE or GETDATE().

- Extraction subset extraction

  Epiphany supports *extraction subset extraction* by reading and recording the current values of these fields at the start of the run and remembering these values for the next run. The values are made available in SQL through the SQL macros described in Appendix B.

- Ignore Timestamps option

  The Ignore Timestamps option, which you can set using EpiCenter Manager's Job dialog box, causes all of the value range expressions to become (1=1), or true, which means that all data is selected without regard to its value range.

- "Priming" value range memory

  The value range memory needs to be "primed" by a trial run whenever you add a new range expression. Basically, the first run causes the memory to start tracking the values for the next run, but all ranges are 1=1 for the current run. The second run causes the "current" values to be read, but all ranges are 1=1 because there are no last values. The third and successive runs have both current and last values, and the range expressions are valid. If you have changed the SQL and are in doubt, make a trial run, which will prime the value memory with any new value expressions.

- Disabling/enabling an expression

  If you remove an expression and later return it, the first run after the expression is back picks up all values since the last time the expression was enabled. If you disable a SQL statement that contains a column value expression and no other SQL uses that same column value, the column value from the last enabled run of that SQL will be used as the last value whenever the SQL is re-enabled.

  Disabled expressions related to dates or timestamps will pick up the last value that any expression found. If all expressions using dates and timestamps are disabled, they will pick up where they left off when they

are again enabled. If any expression using a date or timestamp remains enabled, it "bumps" the date and timestamp memory, and all expressions will pick up the new values when they are again enabled.

- Memory for last values

The "memory" for last values is associated with an extractor and not a SQL statement or a job. If the same extractor is used in different jobs, the jobs will pick up each other's value updates. For example, the Weekend job will see the values updated by the Weekday jobs if they share extractors.

If the same SQL step is used in two extractors, it will have different "last" values for each extractor, which is desirable because the extractors may access different databases that have different highest values and times.

EpiChannel Debugging

The EpiChannel debugging mechanism is thorough and well annotated. If you set a high enough verbosity level, the complete anatomy of the job is revealed in real time at the console, or historically in the log file. The EpiChannel **-v** option controls the verbosity level. (Running **extract** with the help (**-?** or **-h**) option lists and defines these verbosity levels.)

Job Output

You may display EpiChannel output for debugging purposes on a screen or direct it to a file. The text is displayed on the screen in fixed fonts, such as Courier. The number of characters that can be displayed per line is determined by the number of columns. (You may use EpiCenter Manager's Job dialog box to set the job log width to 80 or 132.) The output is designed to be adequately displayed on low-end monitors.

At the top of the debugging screen/log, you will find the names and directory locations of the job's EpiMeta database, as well as its *job log, log file*, and *log database*. (See Figure 2-4 for the start of an output file and Figure 2-5 for the log summary.)

Typically, the job begins with the truncation (deletion) of old staging tables and the clearing of indexes. The processing of each step is displayed sequentially; for example:

```
1.1 Processing step 'PreMfg' as an extractor
```

followed by the extractor's source and destination databases. Steps that are not enabled are numbered, but not shown. In the job output shown in Figure 2-4, Step 1.1 was disabled.

Each operation follows on a separate line. You can identify the type of operation by the letters or symbol(s) that precedes it:

- **St** A set of SQL statements. The members of this group can be either stand-alone SQL, extraction statements (pull/push), or semantic instances (data merges). Steps may be one of these types:

      < Stand-alone SQL applied to the source data store.

      > Stand-alone SQL applied to the destination data store.

      <> Pull/push SQL that transfers data from the source data store to the output data store.

- **SI** Semantic instances.

- **Sy** System calls.

The EpiChannel output provides summary information that is based on the statement type. The summary information is prioritized and displayed in columns to the right of the operation. If there is not enough space, only the highest priority information is given.

For SQL pull/push statements, the output includes the time taken for the pull/push, the number of rows in the table before the operation, the number of rows transferred, and the number of rows in the table at the end of the procedure. If the log file's width is 132, the number of bytes is also shown.

Stand-alone SQL statements have time as their only metric. Semantic instances define their own metrics.

To conserve space, the summary information is labeled by two-digit codes (tokens) that appear to the right of their numeric values. These tokens are defined as follows:

- **Ti**   Time taken, rounded to the nearest second.

- **St**   Rows present in the table at the start.

- **Ex**   Rows extracted.

- **En**   Rows present in the table at the end.

- **UN**   Rows unjoined. These fact table rows contained references to dimension values not present in the EpiMart or in the newly extracted data.

- **PR**   Rows processed; a count of the gross number of rows examined.

- **IN**   Rows inserted; a count of the rows that were successfully moved into the EpiMart. Rows are sometimes omitted because they are earlier than previously recorded data, or because they are redundant to data present elsewhere in the same staging table.

- **MO**   Rows modified. Modifications typically adjust references to missing dimension values such that they become references to the predefined UNKNOWN dimension value.

For example, the lines:

Statement Name	Time	Metric1	Metric2	Metric3
St wb bump the test day				
< wb bump the test day				
<> appl real		0 St	3 Ex	3 En
SI Product	1 TI	3 IN	0 MO	3 PR

indicate the following:

- A set of statements called wb bump the test day was called.

- There is no summary information for this set.

- An execute-only SQL statement called wb bump the test day was run against the source system and took less than a half second to execute, and therefore has no time statistic.

- An extraction statement called appl real inserted 3 rows into a previously empty table and took less than half a second to execute.

- A semantic instance called Product took about a second to execute and processed 3 rows, modifying the contents of none, and inserting 3 rows as data into the EpiMart.

The summary table at the end of the report shows totals for the amount of time each operation took and the number of rows, bytes, and metadata objects that were extracted. Using this information, you can quickly determine if a problem resides in EpiChannel, the semantic instances, or in the extraction SQL statements or system calls.

```
Job testjcb on instance epiwb, InitialLoad
  (Meta DB 'chnCmd' on SQLServer nessie::epiwb_epimeta)
  (Working file 'Job Log' at
    d:\metadata\logfiles\chn_1998-04-01_09-03\testjob.log)
  (Log file 'FileLogging' at
    d:\metadata\logfiles2\chn_1998-04-01_09-03\testjcb.lcg)
  (Log DB 'DBLog_1' on SQLServer nessie::epiwb_epimeta)


========================= Preparing Job =====================================
  All databases exist and can be opened

  Checking tables ...
    Verified existence of 20 tables in data store Epimart

  Truncating ...
    Truncated 20 tables in data store Epimart


========================= Executing Job =====================================

-- 2.2 Processing step 'testjob' as an extractor
        Testdata ==> TestDest_svr
```

Statement Name	Time	Metric 1	Metric 2	Metric 3
St testjob dump tran				
> dump tran on meta				
St testjob extraction stat				
<> cust real		0 St	0 Ex	0 En
<> product real		0 St	0 Ex	0 En
<> territory real		0 St	0 Ex	0 En
<> program real		0 St	0 Ex	0 En
<> appl real		0 St	0 Ex	0 En
St testjob semantic instan				
SI Product	1 Ti	0 IN	0 MO	0 PR
SI Application	1 Ti	0 IN	0 MO	0 PR
SI Customer		0 IN	0 MO	0 PR
SI Territory	1 Ti	0 IN	0 MO	0 PR
SI Program		0 IN	0 MO	0 PR

Figure 2-4 Sample Job Output (Preparation and Execution)

```
St testjob semantic instan |            |            |            |
...[Operations for St testjob are deleted from this sample.]

-- 3.3 Processing step AggBuilder as a system call
   Statement Name          Time        Metric  1   Metric  2   Metric  3
------------------------ -----------  ----------- ----------- -----------
Sy AggBuilder            |      8 Ti|        8 Tm|                |

-- 4.4 Processing step 'HotSwap' as an extractor
       Testdata ==> TestDest_svr

   Statement Name          Time        Metric  1   Metric  2   Metric  3
------------------------ -----------  ----------- ----------- -----------
St Calc Extraction Times --|            |           |           |
  >    Get Customer extractio|           |           |           |
  >    Get Product extraction|           |           |           |
  >    Get max of the maxes  |           |           |           |
St HotSwap -- EpiCenter ind|             |           |           |
  >    Hotswap 2            |            |           |           |

Successfully finished body of job testjob

============================ Committing Job =================================

Label     EntireJob testjob  AggBuilde HotSwap
--------- --------- --------- --------- ---------
Time Totl|       9|         |         8|
Tme Latch|        |         |          |
Tme SysC |       8|         |         8|
Tme Extr |        |         |          |
Tme SI   |      11|       11|          |
Tme AT   |      10|       10|          |
Tme Elks |       5|        5|          |
Tme SQLPa|       3|         |          |
Tme JobR |       4|         |          |
Tme Cnt R|        |         |          |
Extr Rows|       0|        0|          |
Bytes Ext|       0|        0|          |
MetaObjs*|     498|         |          |

Committed Job testjob
```

Figure 2-5 Sample Job Output (Summary)

Error Messages

An EpiChannel error message distinguishes between externally generated errors, such as database errors, and internally created messages, such as a file data store that references a nonexistent directory. Each error includes information about the database/file that caused the error, and which specific part of the code detected the error. Most errors contain information about the operations affected or aborted by the error, the metadata objects that specified these operations, and the program's plan of revised actions. Many errors also suggest corrective action.

There are specific error messages for the following:

- SELECT statements that do not match columns in the metadata. These display mappings indicate which return values were or were not accepted.

- Rows that fail to insert have their complete set of data displayed.

EpiChannel Debugging Levels

EpiChannel's verbose (-v) option allows you to select how much of the SQL displays on the screen during extraction. For a description of each debugging level, run EpiChannel with the -h or -? parameter. The default level shows jobs, steps, SQL statements, semantic instances, and statistics.

The lowest level shows only jobs and errors. Increasing levels show the SQL pre- and post-expansion of macros, templates, and blocks within semantic instances, or even each individual row returned from a SELECT statement. At the highest levels of verbosity, execution pauses between SQL statements and between returned rows, which allows you to trace behavior precisely.

You may even use EpiChannel interactively to alter the data fetched by a SELECT statement before the row is forwarded for insertion. See *Setting Breakpoints* below.

Note: To turn off verbosity, select -4.

Setting Breakpoints

You may set breakpoints on SQL statements that change the debug level *before* the SQL is issued. These breaks can also be associated with particular rows in the return set. For example, it is possible to set a breakpoint just before row 3021 of the *Orders table extraction* SQL statement.

You can use the EpiCenter Manager's SQL Statement dialog box to set breakpoints (see Figure 3-19, page 98). Follow these steps:

1. Select the debug level. These levels correspond to the verbosity levels on the **extract** command line (or in the Execute Job dialog box shown in Figure 3-22, page 105).

2. Select the row number at which the debug level you selected above will go into effect; this is the row number immediately *before* the SQL statement is executed. If the row number is not empty, and the level is "row pause" or higher, the pull/push loop stops fetching before inserting that particular row and waits for permission to proceed. You may alter the data during this pause.

When you set a breakpoint, it stays set for the remainder of the job. This prevents you from having to set a debug level on entire groups, but it does necessitate your setting the breakpoint back to the original value later in the job flow if you do not want the entire job debugged (and remembering to remove this breakpoint).

All SQL issued against the source, destination, or metadata databases will be logged, if specified by the debug levels.

Trial Runs

You can run EpiChannel in trial-run mode, in which SQL is not actually issued against the source or destination systems. This option is recommended for testing the structure of an extraction job.

Maximum limits can be set on the number of rows to be fetched from any SELECT statement. When used, these limits essentially cause the SQL to be checked for syntax on a small selection set.

The execution time for all SQL executed is tracked and logged.

EpiChannel Output

You must provide EpiChannel with one and only one file data store with the role of *working directory*. A working directories is a log directory that is also used as the location for temporary files associated with the job.

You may, however, have any number of file or database data stores that have the role of *log*. The locations referenced by data stores with these roles are known as log directories and log databases, depending on their data store type. You may use more than one log database and more than one log file. In this case, all logging activity is duplicated to all of the listed data stores. Having a backup of each kind of data store is recommended if you require logging information to be constantly available for the administration of your system, or if data stores could possibly go off-line or be destroyed.

You may also use multiple listings so that one log is local to the host and one is remote. For example, during installation and early testing, you might direct one database log to a database hosted by Epiphany for the monitoring of your site's activities.

> *Note:* The EpiCenter Manager default data stores may be sufficient for your site.

Log and Working Directories

The location specified for a *log* or *working directory* is a parent directory under which a new subdirectory will be created for every job. The subdirectories are named based on a timestamp from the start of the EpiChannel run. If there is already a subdirectory of that name, then a suffix is added to make the name unique to that directory. Under the subdirectory is a file called *jobname.log* with the log and informational messages from that job.

Log Files versus Log Databases

EpiChannel can log to both files and databases. Log files provide detailed information regarding a single job and log databases provide complementary but less detailed status and performance information about all jobs. Log files help you to a determine the success of a job and suggest corrective action. Log databases enable you to analyze the behavior of jobs over time in order to detect troublesome areas, or areas of degrading performance.

> *Note:* A log database is not a log device. A log device is a construct in SQLServer that logs database transactional activity.

Monitoring Jobs

You may use the NT Performance Monitor to determine the current status of the database extraction process; for example, you can find out if the cause of a longer than normal extraction process is the result of network performance problems, or simply more rows being extracted and processed.

You may use one or more of these measures to monitor job behavior:

* The total number of jobs started. Use this to see transitional boundaries in the work being done.

* The total number of extraction nodes started. Use this to see transitional boundaries in the work being done.

* The total number of rows transferred in a pull/push operation. This value is reset with each extraction statement.

- The total number of bytes transferred in a pull/push operation. This value is reset with each extraction statement.

- The number of rows committed. This value is reset with each extraction statement.

- The rate of row transfer in a pull/push operation (rows per second).

- The rate of byte transfer in a pull/push operation (bytes per second).

- Total number of semantic instance blocks called. Use this to see transitional boundaries in the work being done.

- The rate at which semantic instance blocks are called. This shows the percentage of the clock time spent in semantic instance calls.

- The number of meta objects in the job, which is a measure of the complexity of the workflow.

You may combine these metrics with the other measures available through the Performance Monitor. For example, the bytes per second can be displayed along with the TCP packets per second, the average length of the disk queue, and SQLServer-specific measures.

Running the Performance Monitor

See *Setting Up NT Performance Monitoring for Extraction (Optional)* on page 222 for instructions on how to set up the Performance Monitor for EpiChannel usage.

After setting up the Performance Monitor, you can monitor any EpiChannel jobs that are run on this machine. Follow these steps:

1. Start the NT Performance Monitor. (It is located in the *Start\ Programs\Administrative Tools* menu on most machines.)

2. Click the ADD or plus (+) button in the Performance Monitors' window.

3. Select Epiphany Channels from the object list.

4. Select the measures you want to monitor from the counter list.

Mirroring: A and B Tables

The Epiphany system uses the metadata definitions of facts and dimensions to construct the physical EpiMart tables. Table naming conventions indicate how the table is used. For instance, if a base dimension table named Customer has been defined via EpiCenter Manager, the system constructs these tables:

- Customer_O_A

- Customer_O_B

- CustomerMap_A

- CustomerMap_B

- CustomerStage

The _O, Map, or Stage suffixes are needed for the normal operation of EpiMart. Note that the Customer_O and Map tables appear twice in this list: once each with a suffix of A and B. The purpose of this table "mirroring" is to allow extractions to occur without disrupting the live usage of a system.

At any time, either the letter A or B is "active," meaning that all end-user queries will operate against tables with that suffix. This setting is determined by an option in the EpiCenter Manager's Configuration dialog box (described in Appendix C) called current_datamart. If this option is set to A, then all end-user queries will be routed against tables that end in A (and all of the B tables will lie dormant). The B tables contain data that is one extraction older than the A tables.

During an extraction, the data in the active tables (A in this example) is never changed. Instead, the B tables are constructed by combining the A tables' data with any newly extracted data from the staging tables. In other words, the dormant set of tables is rebuilt every extraction with the latest data. Not only does extraction of the base tables (those that end with _O) occur in the dormant set, but aggregation is also performed on the B tables. In this way, end users

can continue to query against the A tables, without knowing that extraction and aggregation are occurring simultaneously against the B tables.

If extraction finishes successfully, then the value of the configuration setting *current_datamart* is toggled to be the dormant letter (B in this case). As soon as the Epiphany Application Server (described in Chapter 5) is notified that the data extraction was successful, all end-user queries will be directed against the B tables. In this way, the A tables, which contain one less extraction's worth of data, become the dormant set. The next time EpiChannel is run, the A tables will be recreated, and the letter A will once again become active.

Note the following:

- Aggregate building is always performed against the set of tables not equal to the *current_datamart* setting.

- In order to completely re-extract an entire system, the set of tables whose suffixes equal the *current_datamart* setting must be truncated. (This, however, affects current end users of the system and will be resolved by the introduction of a new semantic type that allows complete re-extraction without truncating any EpiMart tables.)

CHAPTER THREE

EpiCenter Manager

EpiCenter Manager is Epiphany's versatile program for configuring, administering, re-configuring, and updating your database. You will use its graphical user interface to define your EpiMeta, or database table schema, as well as the jobs needed for data extraction. You will also use EpiCenter Manager to perform administrative tasks, such as restricting access to ticksheets and sending e-mail notification of job status. When your schema changes incrementally, EpiCenter Manager allows you to update it "on the fly," without rebuilding the entire EpiMart.

This chapter describes how to use EpiCenter Manager to set up your databases for optimal extraction of your source data, to define jobs, and to administer your system. The following basic concepts are also discussed:

- Uniform Treatment of Time

- Uniform Transactional Data

- Epiphany's Adaptive Architecture

Planning Your EpiCenters

EpiCenter Manager is a program for experienced database administrators who are familiar with their site's database and database needs. Because incorrectly updating schema can corrupt an existing database, it is important that you understand Epiphany system concepts and how to use

EpiCenter Manager in a way that is appropriate for your site. Before using EpiCenter Manager, be sure to read Chapters 1 and 2 of this *Guide*, which describe the Epiphany system and the database extraction process. Many of the key terms, such as EpiCenter, EpiMeta, and EpiMart are described in these chapters.

Before you configure an EpiCenter, your site needs to analyze its business model to determine:

- How many EpiCenters are appropriate.

  An EpiCenter is a construct for organizing your metadata into a database called EpiMeta that defines a single EpiMart database. EpiMart is where the data extracted from your source systems resides. If your system has adequate memory and storage (see Appendix A, *Installation*), and your business model warrants it, you may set up multiple EpiCenters. For example, you may have one for development and one for production.

- The number of constellations within each EpiCenter.

  This includes the fact tables and dimension tables (and aggregates) that should be logically organized into a constellation. Although every EpiCenter has at least one constellation, some EpiCenters will have multiple constellations. The number depends on the site's business model; for example, one for each business process.

- The extraction process for your site.

  Analyze the kinds of jobs and job steps needed; for example, you may create job flow chart(s). After you design the system at this higher level, you will be better able to—

  - Identify the extractors (SQL statements and semantic instances) needed to extract information from the source system(s) and write these extractors.

  - Define external tables.

  - Identify requisite system-call commands.

Getting Started

Follow these steps to start EpiCenter Manager and register a database server:

1. Complete the installation as described in Appendix A, *Installation*.

   The installation procedure includes creating new, empty databases on the server for the EpiMeta and EpiMart databases and installing the Epiphany software. Although you create two databases, the EpiMeta is the one you work with using EpiCenter Manager. The EpiMeta defines the database tables for your data warehouse (the EpiMart).

2. Start EpiCenter Manager.

   During installation this program executable (**EpiMgr.exe**) is placed in the Epiphany directory by default (*C:\Program Files\Epiphany*) and can be selected from the *Start* menu:
   Programs\Epiphany\instance_name\EpiCenter Manager.

   The EpiCenter Enterprise Manager window is displayed.

   At the top level is the Servers icon. Follow these steps to register your server:

1. Choose Register from the Server menu. The Server Properties dialog box (Figure 3-1, page 60) is displayed in the window.

2. Select the Server Type from the drop-down list and enter the database server's name, and the username and password for this server.

   Your database server machine icon and the EpiCenters folder (or directory) appear in the hierarchical tree structure (see Figure 3-2, page 61). This server icon represents the server where the EpiMart and EpiMeta databases for this EpiCenter reside.
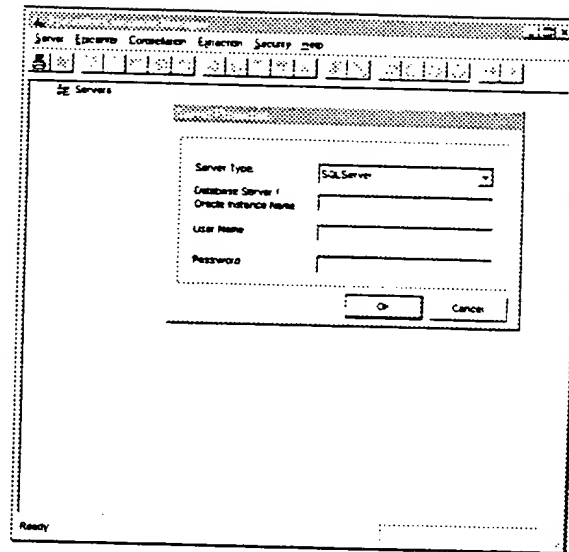
Figure 3-1 Registering the Database Server

3. Click Cancel to close the empty Choose EpiCenter dialog box.

Later you may use this dialog box to select and open an already initialized EpiCenter as described in *Working with an Existing EpiCenter*, page 66.

When you register a server for the first time, after connecting to that machine, EpiCenter Manager displays the Register EpiCenter dialog box. Close this dialog to continue with the initialization.

Right-
click the
EpiCenters
folder
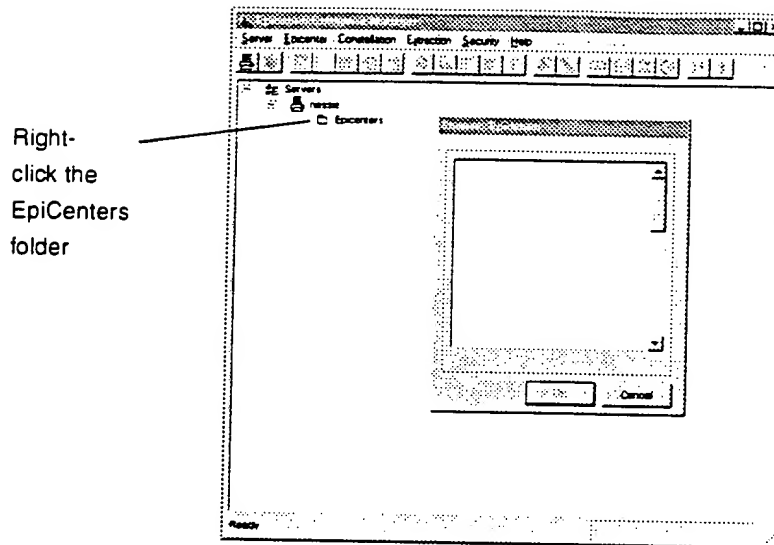
Figure 3-2 EpiCenter Enterprise Manager Window

Follow these steps to initialize your EpiCenter:

1. Right-click the EpiCenters folder in the EpiCenter Manager Enterprise window (see Figure 3-2) and select Initialize EpiCenter from the pop-up menu. The Initialize EpiCenter dialog box is displayed (see Figure 3-3, page 62).

2. In the EpiMeta Name drop-down list, select the name of the new, empty database for the metadata that you created during installation (as described in Appendix A, *Installation*).

3. In the EpiMart Name drop-down list, select the name of the new, empty database for your customer data that you created during installation.

4. Click Build to start building a generic EpiMeta database of the size you specified. The status of the build is displayed in the lower part of the (now expanded) Initialize EpiCenter dialog box.
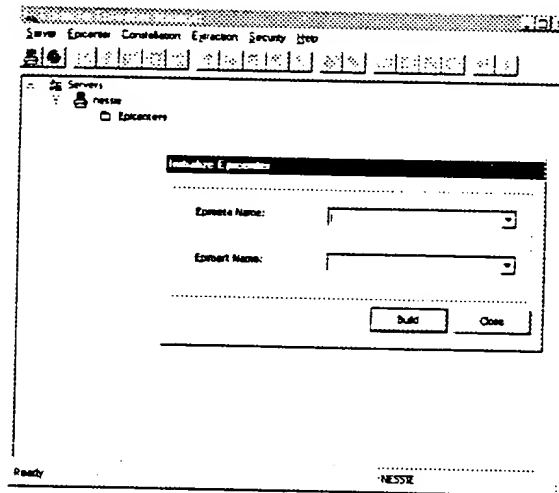
Figure 3-3 Initializing Your EpiCenter

The new EpiCenter icon is added to the directory tree with folders for Base Dimensions, Constellations, Extraction, and Security (see Figure 3-4). Note that the icons in the window are arranged in an Explorer-like hierarchical tree.
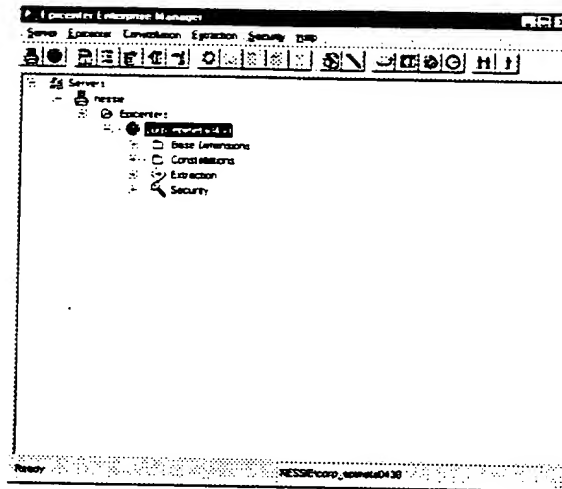
Figure 3-4 EpiCenter Enterprise Manager Window

Next, the Date Dimension dialog box is displayed. You will need to populate the actual, physical date dimension table. This is a special base dimension table supplied by Epiphany that is used for storing all attributes related to time. All fact tables in an EpiCenter receive a foreign key (called *date_key*) to this table.

Note: If you receive the message: *Cannot open the datamart in single-user mode*, close any other applications that are currently using this database, including the SQLServer Manager in which the database was created. For more information about this error message, see page 68.

To populate the date dimension table:

1. Choose Populate Date Dimension from the EpiCenter menu.

2. In the Date Dimension dialog box (Figure 3-5), select the values for the beginning and ending year of the EpiMart and your calendar type and start day. The date range of the EpiMart should be at least as large as any

dates that are found in the data you will be extracting. These values are defined in the Configuration dialog box (choose Configuration from the EpiCenter menu to open it).

In order for users to get the best results when forecasting trends using Relevance, you need to build the date dimension as far into the future as you plan to forecast. (Currently, the maximum prediction is three years past the last date that has recorded data.) If the date dimension is not built out far enough, the user will receive columns with names such as *Dec 1999, Second Next, Third Next* instead of *Dec 1999, Dec 2000, Dec 2001.*

Note: Quarters 4-4-5 represents the 13-week-per-quarter calendar in which the months in the quarter are defined as consisting of 4 weeks, 4 weeks, and 5 weeks.

3. Click Build. The EpiMeta database will be populated with a base list of dates.

For descriptions of these date fields, see *Configuration,* page 243 in Appendix C and Appendix D, *Date Dimension Fields.*

Figure 3-5 Date Dimension Dialog Box

A discussion of how the Epiphany system treats time follows. Please read this section and read the introductory material in *The EpiCenter Manager Window* and *Working with EpiCenter Manager*, page 67 before beginning to set up your EpiCenter as described in *Setting Up an EpiCenter*, page 69.

Basic Concept: The Uniform Treatment of Time

In many Enterprise Relationship Management (ERM) applications, the time on which a fact occurs is a crucial dimension. It can be complicated, however, to create the semantics of calendars and other date attributes for a new data warehouse. Epiphany provides a time dimension via the special table *Date_0*, along with EpiCenter Manager, for populating the date dimension with the parameters that make sense for each implementation. Each row in this table represents a single day and the attributes associated with that day.

Every fact table in EpiMart automatically contains a foreign key to this special date dimension. Many of the fact semantic templates (the generic programs that perform business transformations on extracted data) depend on the presence of this field. By joining the fact table to the date dimension, Epiphany's ERM applications are able to answer questions of the form:

- Which facts occurred this quarter?

- What is my weekly backlog for the year?

- What products were bought last month?

The queries that are constructed for these purposes do not perform date arithmetic, such as:

```
SELECT * FROM MyFact
WHERE date_key > '6/1/1998' and date_key < '7/1/1998'
```

Queries of the above form become unwieldy because of the non-uniformity of the calendar with respect to the number of days in the month, leap years, weekly overlaps with months, and so on. Instead, determining which days belong to which weeks, months, quarters, or years is done once when the date dimension is populated. This table is then fully enumerated with all values of interest for the EpiCenter, allowing queries of the form:

```
SELECT * FROM MyFact, Date_0 WHERE MyFact.date_key =Date_0.date_key
AND cq_and_cy_name = 'Jun 1998'
```

Queries of this form can easily take advantage of RDBMS indexes.

Epiphany allows for specification of Calendar fiscal quarters, as well as 13-week manufacturing calendars (4-4-5 calendars), in which a quarter always starts on the same day of the week. The date dimension can also be configured to specify which weekdays start and end a business week (for example, Sunday through Saturday versus Monday through Sunday).

Working with an Existing EpiCenter

If an EpiCenter has already been initialized, you may register it as follows:

1. In the EpiCenter Enterprise Manager window, right-click the EpiCenters folder icon and select Register from the pop-up menu.

2. In the Choose EpiCenter dialog box, select the EpiMeta for the EpiCenter.

The EpiCenter's hierarchical tree is displayed in the EpiCenter Manager Enterprise window. You may modify this EpiCenter as appropriate for your site by following the instructions given in this chapter for those items you wish to change.

You can unregister an EpiCenter by right-clicking the EpiCenters folder icon (see Figure 3-2, page 61) and selecting Unregister from the pop-up menu. The EpiCenter's hierarchical tree is removed from the window.

The EpiCenter Manager Window

Similar to other Windows applications, you may invoke EpiCenter Manager commands by using the main menu, toolbar icons, or by right-clicking icons in the window and selecting commands from a pop-up menu. Before you start configuring your EpiCenter, familiarize yourself with the main menu commands and the toolbar icons. The main menu commands are organized by the main EpiCenter Manager functions: Server, EpiCenter, Constellation, Extraction, Security, and Help. These menus are contextual and their commands are accessible when you are working within their function. For example, the Constellation commands are available when you working with a Constellation, but may be unavailable in other contexts.

EpiCenter toolbar icons provide access to special functions such as truncating tables, or displaying dialog boxes you use on a regular basis, such as the new Job dialog box and dialog boxes for defining facts, dimensions, external tables, and users and groups. A toolbar icon must be activated (highlighted) for you to use it. To activate a toolbar icon, select a related icon in the directory tree.

Working with EpiCenter Manager

The EpiCenter Manager window is organized in a hierarchical tree structure in which you right-click a plus or minus sign to expand or collapse a directory, as in Windows Explorer. In general, you will work down the tree structure in the window, right-clicking folders or icons to display a contextual pop-up menu, or you may prefer to use the equivalent main menu or toolbar commands. The instructions in this chapter use both ways interchangeably.

Double-clicking an empty folder icon opens a dialog box in which you can define a new item, such as a new constellation. If this item has already been created, double-clicking expands or collapses the directory tree. Double-clicking an existing icon, such as the Default Job icon, opens its dialog box.

The following applies to all of your work with EpiCenter Manager:

- Descriptions in dialog boxes

  The Epiphany system does not utilize the descriptions in the Description text area of the EpiCenter Manager dialog boxes. This description is for your documentation purposes only.

- Updating and refreshing

  You may click the Update button in a dialog box to change the EpiMeta description of an existing row of data in the database "on the fly."

  If multiple users are running EpiCenter Manager, metadata changes made by one user are visible to others only after that user issues the Refresh command.

- Unregistering and deleting items

  You can unregister servers and EpiCenters by right-clicking their icons and selecting Unregister from the pop-up menu. You can delete an item in the EpiCenter Enterprise Manager window, such as a base dimension or constellation, by right-clicking its icon and selecting Delete, or by selecting it and pressing the Delete key.

- An explanation of the error message: *Cannot open the datamart in single-user mode* follows:

  Whenever the **EpiMgr** program operates on EpiMart (adding tables, populating the date dimension, and so forth), it attempts to put the database in single-user mode. This is to make sure that no one is running a query when the operation is being performed. This error message results if another connection is open to the same database. To solve this problem, close any other applications that are currently using this database, including the SQLServer Manager in which the database was created.

Setting Up an EpiCenter

The purpose of setting up an EpiCenter is to define your site's metadata, or schema. After defining it, you will use the Generate Schema command in the EpiCenter menu to write these definitions to the EpiMart database. The EpiMeta database points to your EpiMart (where your extracted data is stored).

The instructions in the rest of this chapter describe how to set up a single EpiCenter with one constellation. Some EpiCenters, however, will have more than one constellation. To create additional ones, simply repeat the procedure you used to create your first constellation.

Configuration

The Configuration dialog box contains general settings, lists transaction types, and defines how measure units are displayed. (Choose Configuration from the EpiCenter menu to open this dialog box.) Transaction types distinguish rows with different interpretations in the same fact table. For instance, an Order fact table might contain both a BOOK and SHIP transaction type, differentiated by the value of the column *transtype_key*. Other than your mail password, the information in this dialog box should need little, if any modification. (Most of these settings can be set using other EpiCenter Manager dialog boxes.) See Appendix C for more information.

Base Dimension Tables

The base dimension tables are physical dimension tables that can be used multiple times within a constellation or across constellations. All EpiCenters are created with two default base dimension tables: Date and Transtype.

A base dimension table consists of dimension columns. These are columns in a physical dimension base table that hold attributes extracted from the source system.

Use the Base Dimension dialog box to—

* define the EpiCenter's base dimension tables, such as Product, CostCenter, Account, Subaccount, or Project.

- configure the dimension aggregates used for aggregate building.

To define a new base dimension table:

Right-click the Base Dimensions folder and select New Base Dimension from the pop-up menu. The Base Dimension dialog box (Figure 3-6) is displayed.

This dialog box has three tabs: General, Column Sets, and Built Aggregates.
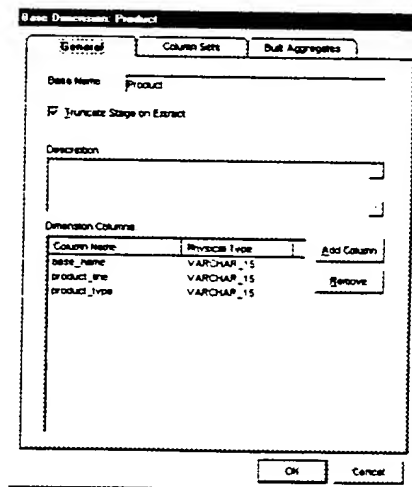
Figure 3-6 Base Dimension Dialog Box: General Tab

Use the General tab to define the base dimension table:

1. Enter the name of the base dimension table, such as *Product*. This is the name of the physical table in the database; underscore zero (_0) will be added to this name to form the base table.

2. By default, the data in the base dimension staging tables is deleted (truncated) after extraction.

    In most cases, you will want to truncate staging tables. You may, however, want to de-select Truncate Stage on Extract when an execute-only SQL statement truncates the staging tables based on criteria you specify for the extraction process. For example, this SQL could delete all records whose foreign keys matched, or all records over seven-days-old.

Alternatively, you may use a single dialog box to define all tables within a constellation that will be truncated prior to extraction (see *Truncating Tables*, page 107).

3. Click Add Column to enter one of the table's column names.

4. In the Dimension Column dialog box, enter a name and a description for the column.

5. Select the physical type from the drop-down list. The physical type you select is replaced by its associated database type. The translation of physical type to database type depends on your RDBMS platform. See Appendix E, *Physical Type Values*, for descriptions of these physical types.

6. Enter a default value for the column.

   By default, each dimension column assumes the literal value UNKNOWN for each of its textual attributes. Leave this field blank to accept the default. To override this value so that end-user queries will return a different value, enter a literal string (without quotes). Your string will be used instead of UNKNOWN. See *The UNKNOWN Dimension Row*, page 34.

Defining Dimension Aggregates

A dimension aggregate represents a dimension table in which one or more of the columns have been removed, and the rows have been collapsed onto one another to remove duplicates. (In the example given in *Aggregate Building*, page 28, if Customer Name is removed from the base Customer table, only two distinct Regions are found in the table, so only two rows are needed in the Region aggregate table.)

Use the Columns Sets tab of the Base Dimension dialog box (Figure 3-7, page 73) to define which aggregates will be built the next time that the Aggbuilder program (**agg.exe**) is executed.

Column sets are subsets of the full list of dimension columns for the associated dimension base table. Each column set represents a *potential* aggregate to be built: the Aggbuilder program does not build all defined columns sets, it builds only those columns sets that are included in fact aggregates (see *Aggregation and Aggregate Grouping*, page 81).

To define dimension aggregates, follow these steps:

> *Note:* The data you enter in the Column Sets tab is used by the Aggregate Building process the next time **agg.exe** is run. Your changes do *not* affect the current EpiMart tables.

1. Enter the name of the column set in the text box, and click New.

   This name is a logical identifier only, and is not used during the building of the Actual table in the EpiMart. Instead, EpiCenter Manager will append unique numbers after the base dimension table name.

2. Select or de-select Default.

   Selecting Default for this column set affects the behavior of the default aggregate group that exists in each constellation (see *The Default Aggregate Group*, page 84).

3. Add column names for this set by clicking New. Select columns from the list of columns for this base dimension table.
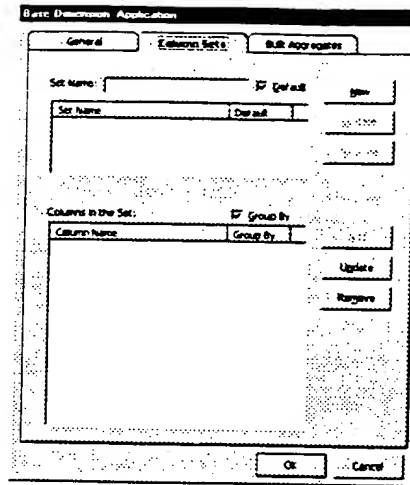
Figure 3-7 Base Dimension Dialog Box: Column Sets Tab

4. Select or de-select Group By.

When including a column in a set, you may specify whether that column is to be used in the GROUP BY clause of the SQL statement that generates the aggregate. Normally, you should select all Group By flags.

Most DBMS's, however, have a limitation of 16 columns in a GROUP BY clause. If a column's value is determined by another column already flagged as Group By, then the former column can be left out of the GROUP BY clause without affecting the result set.

For example, consider a dimension called Product which has, among other things, both a category code and a category name. If each code has a unique associated name, then the code and name should be included in the same set because any aggregation on one is an aggregation on the other. Because the two columns change together, there is no point to group by *category_number* and *category_name* because either group by produces the same results. One of these columns can be left out of the group-by list.

5. Add additional (multiple) column sets by repeating the above steps.

   Multiple dimension column sets are used when a base dimension table with many columns can be aggregated in different ways. Generally, the fewer columns included within a column set, the smaller the resulting dimension aggregate, and the faster queries that use fact aggregates joined to that dimension aggregate will respond.

   For instance, if a customer dimension has several fields related to customer geography (City, State, Zip, and so forth) and several other fields related to demographics (age, marital status, years of education), then EpiCenter Manager can choose two or more dimension column sets (which can overlap) for maximum coverage of queries. At query time, Epiphany's aggregate navigation mechanism will choose the smallest aggregate that satisfies the user's request.

   Thus it is beneficial at query time to have built many small, highly specialized aggregates. The penalty for building many dimension aggregates is that it takes longer to build aggregates, and requires more disk space.

   Note: A small change in the number of dimension aggregates can cause a large change in the number of fact aggregates. The Aggbuilder program builds only those column sets included in fact aggregates (see *Aggregation and Aggregate Grouping,* page 81).

Dimension Aggregate Browsing

While configuring your Epiphany system, you may wish to browse the list of aggregates that Aggbuilder has built (according to the instructions in the Aggregate Group dialog box described in *Aggregation and Aggregate Grouping,* page 81). You can use the Built Aggregates tab of the Base Dimension dialog box to browse the aggregates for either the A or B set of tables.

EpiCenter Manager uses a mirroring structure in which all EpiMart tables are in the database twice, once for each of the _A and _B tables. Only one set of tables, however, is active at a given time. See *Mirroring: A and B Tables*, page 55 for more information.
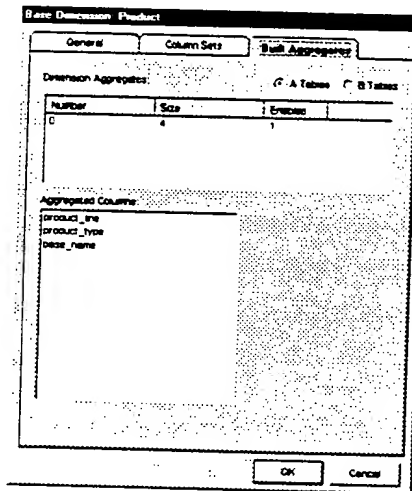
Figure 3-8 Base Dimension Dialog Box: Built Aggregates Tab

Setting Up a Constellation

Within an EpiCenter, constellations serve to group fact tables that have similar dimensions. Instructions are given here for setting up a single constellation. To set up another constellation, repeat the entire series of steps in this section.

To set up a constellation:

1. Choose New Constellation from the EpiCenter menu.

2. Enter the new constellation name and any description.

The New Constellation icon appears in the Constellation folder. It has these sub-folders arranged in alphabetical order: Aggregates, Degenerate Dimensions, Dimension Roles, Facts, Measures, and Ticksheets.

Although you can usually proceed down the tree as you define your EpiCenter, you need to define dimensions and facts before aggregates. Follow the sequence given below.

Defining Dimension Roles

A base dimension table may have a superset of data that can be used for different purposes. For example, a Customer base dimension table may contain Bill-to and Ship-to data. Dimension roles enable base dimension tables to be used more than once in a constellation.

A dimension role indicates the single usage (such as Bill-to data only) of a base dimension table by all of the fact tables within a constellation. A row in a fact table may have several foreign keys that point to the same base dimension table, but the role usage may differ (for example, some relate to Bill-to and others to Ship-to data). The dimension role maps the fact foreign keys to the correct base dimension table.

To define roles for this dimension table:

1. Right-click the Dimension Role folder and select New Dimension Role. The Dimension Role dialog box is displayed.

2. Select the base dimension name from the drop-down list.

3. Enter the Dimension Role Name and a description, and click OK. The dimension role icon is added to the tree.

4. Repeat the above steps for any other base dimension tables that have multiple roles.

Every dimension role must have a parent base dimension table. If there is no base dimension for this role, click New in the Dimension Role dialog box. Use the Base Dimension dialog box that displays to create a new base dimension.

Degenerate Dimensions

Certain numbers in fact tables such as order, invoice, and bill of lading numbers have foreign keys with no corresponding dimension table. The only data of importance is the number itself, all of the other information of value has been extracted into other dimension tables. These numbers are called degenerate dimensions. A degenerate dimension is part of a constellation, and all facts in the constellation inherit it. If you use degenerate dimensions, you need to inform the system of their existence.

To define a degenerate dimension:

Choose New Degenerate Dimension from the Constellation menu. Enter the degenerate dimension name and a description in the Degenerate Dimension dialog box.

Defining Fact Tables

A fact table is a physical table that holds numeric data in its columns. It also represents the intersection of a series of dimension keys. A fact table consists of dimension role foreign keys, degenerate dimension keys, and fact columns.

To define a fact table:

1. Right-click the Facts folder icon, and select New Fact. The Fact Table dialog box (Figure 3-9) is displayed. This dialog box has three tabs: General, Custom Index, and Built Aggregates.

2. In the General tab, enter the fact table's name and text that describes this fact table.

3. By default, the data in the fact staging tables is deleted (truncated) after extraction. In most cases, you will accept the default.

4. By default, the *Build aggregates for this Fact* check box is selected. De-select it to turn off aggregate building.
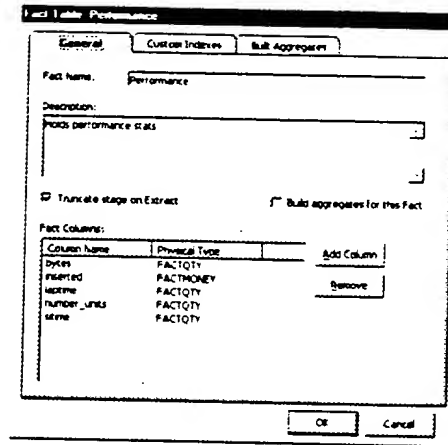
Figure 3-9 Fact Table Dialog Box

Defining Fact Columns

Fact columns contain the actual customer numeric data; such as, *net_price* or *number_units*. You will use the General tab of the Fact Table dialog box to define a fact column:

1. Click Add Column. The Fact Column dialog box (Figure 3-10) is displayed.
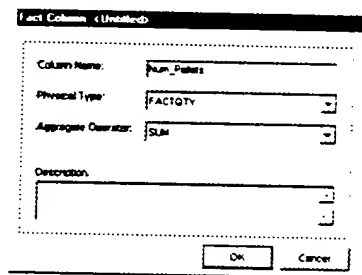
Figure 3-10 Fact Column Dialog Box

2. Enter the column name and select the physical type from the drop-down list. The physical type you select is replaced by its associated database type. The translation of physical type to database type depends on your RDBMS platform. See Appendix E for descriptions of these physical types.

3. Select the aggregate operator from the drop-down list. Aggregate operators determine how the Aggbuilder program calculates the facts. (For this release, SUM is the only option.)

4. Enter a description, and click OK to return to the Fact Table dialog box. The new fact column appears in the listbox.

5. Repeat the above steps to define another column.

To remove a column name, select it and click Remove.

Custom Indexes

Custom indexes can dramatically improve query performance. See *Custom Fact Indexing*, page 33 for background information.

Use the Custom Indexes tab of the Fact Table dialog box (Figure 3-11) to define indexes.

To define a custom index:

1. Enter the name of a custom Index in the text box and click New. The index name is added to the list.

2. For *Dimensions in the index*, select the index in the list and click Add Dimensions. The Choose Dimension dialog box is displayed.

3. Select one of the dimension roles from the list.

4. Click OK to add this dimension to the list.
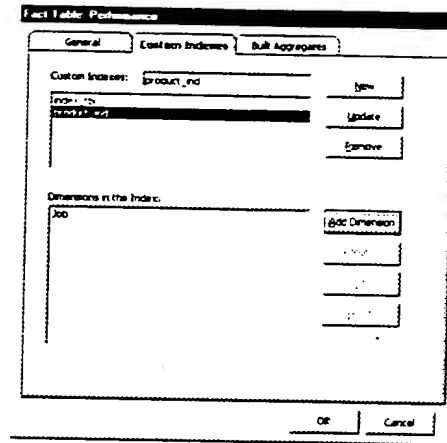
Figure 3-11 Fact Table Dialog Box: Custom Indexes Tab

Fact Aggregate Browsing

While configuring your Epiphany system, you may wish to browse the list of built fact aggregates to determine which tables are actually available for the query machinery. (See *Aggregation and Aggregate Grouping,* page 81 for more information.)

Use the Built Aggregates tab of the Fact Table dialog box (Figure 3-12) for this purpose. As with dimension aggregate browsing, you can browse either the A or B set of tables, as appropriate. Follow these steps:

1. Select a dimension role from the Role panel and then select a filter from drop-down list. (Make one selection per dimension role.) The filter options are described in Table 3-1 on page 84.

   Filtering specifies that a dimension role—

   • should not be included in the aggregate.

   • should be included as the base table.

- should be included as a specific dimension aggregate (corresponding to a dimension column set).

2. Fact aggregates appear in the Fact Aggregates listbox.

   Clicking a row in this listbox displays the following in the Aggregate Contents panel of the tab: the dimension roles that are included for this fact aggregate, and the dimension aggregates (or the base table) that the fact aggregate joins to.
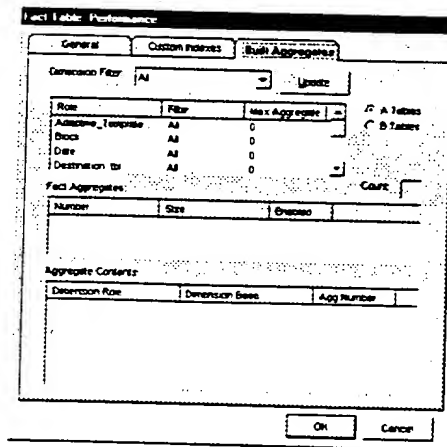
Figure 3-12 Fact Table Dialog Box: Built Aggregates Tab

Aggregation and Aggregate Grouping

Aggregates are facts that the system will pre-calculate for a group of dimensions. (These dimensional groupings are determined by the groups you set up for a base dimension table.) An *aggregate group* is a set of instructions (a metadata definition) that tells Aggbuilder which aggregates to build on one or more fact tables. The fact aggregates that are actually built are determined by a combinatorial expansion of the instructions in the aggregate group.

Unlike dimension aggregates, fact aggregates are not fully enumerated through the use of EpiCenter Manager. The reason for this is simple: while base dimensions typically have only a few aggregates, facts usually have many.

EpiCenter Manager simplifies aggregation by providing aggregate groups that you define using the Aggregate Group dialog box (Figure 3-13, page 85 shows this dialog box for the Default Aggregate Group). For each aggregate group, you will specify the facts in the constellation that share it. (An aggregate group applies to a single constellation.) If two or more facts share an aggregate group, then the Aggbuilder program will follow the same set of instructions for each of the facts.

To open an existing Aggregate Group dialog box, double-click it. (To open a new Aggregate Group dialog box, expand the Constellation tree and right-click the Aggregates folder. Select New Aggregate from the pop-up menu.)

Use the General tab of the Aggregate Group dialog box to define an aggregate group:

1. Enter an aggregate group name (and description).

2. Select Enabled for Aggbuilder for the Aggbuilder program to use this grouping.

3. Select the facts that share this aggregate. Click Add Fact and select facts from the Choose dialog box.

Use the Definition tab of the Aggregate Group dialog box to configure the dimensionality:

1. Click the Add Dim button.

2. Select the dimension roles in this constellation (to be considered for these aggregates) from the Choose dialog box.

   If a dimension role is not included in the aggregate group, then *all* of the aggregates that result from this aggregate group will not include that dimension role.

3. Select a dimension role (in the Dimension Settings listbox) and click Add Set to configure the Other Column Sets aggregate type for this dimension (see Table 3-1).

Dimension aggregates are built during the aggregation process. Each column set for a base dimension will become a dimension aggregate, as long as that column set is included in some aggregate group in the Other Column Sets list.

When a dimension role is included in an aggregate group, you need to specify the possible ways that the dimension role can be built into the aggregates defined by this group. To illustrate this point, assume that you define an aggregate group that consists of the following dimension roles:

Dimension Role	Number of Possibilities
Customer	3 (Not included, Set 1, Set 2)
Product	3 (Set 1, Set 2, Set 3)
SalesPerson	2 (Not included, Base)
Date	5 (Not included, Sets 1-4)

This aggregate produces 3 . 3 . 2 . 5, which equals 90 aggregates for each fact table included in the aggregate group. Aggbuilder will assign a unique number to each of these aggregates, and the resulting table that is built in EpiMart will be *fact table name_#*.

The following table summarizes the choices for inclusion of a dimension role in an aggregate group.

Table 3-1 Dimension Roles for Aggregation

Inclusion	Description
All	The dimension role can either be excluded from an aggregate, have its base table included, or have one of the Other Column Sets included.
NoneOrOthers	The dimension role can either be excluded from an aggregate, or have one of the Other Column Sets included.
BaseOnly	The dimension role will always be included in all aggregates as the Base Dimension Table.
OthersOnly	The dimension role will be included only as one of the Other Column Sets.
BaseOrOthers	The dimension role can either have its base table included, or have one of the Other Column Sets included.
NoneOrBase	The dimension role can either be excluded in an aggregate, or have its base table included.

The Default Aggregate Group

Each constellation has a default aggregate group that has special semantics to make the configuration of aggregates simpler. You cannot delete this group, although you may disable it.
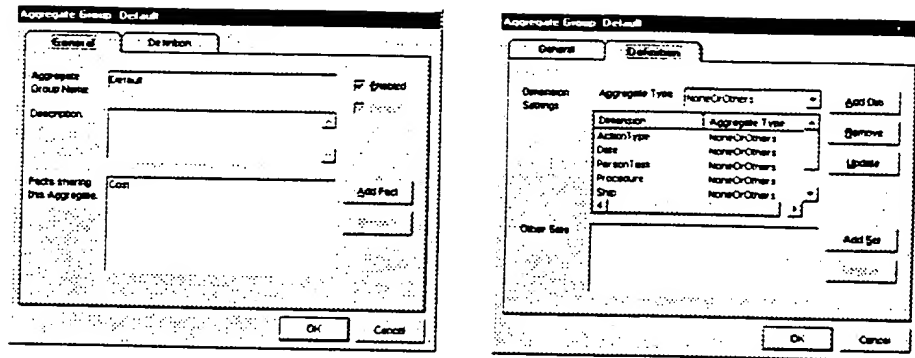
Figure 3-13 Default Aggregate Group Dialog Box

The date dimension is automatically included in this group, since many end-user queries involve time. Date is included as a NoneOrOthers dimension role, and the Other Column Sets listbox is determined by which column sets of the Date Dimension dialog box are set to Default.

When new dimension roles are added to the constellation, the default aggregate group is automatically modified to include those roles in a NoneOrOthers manner. The reason for excluding the base tables from the default aggregate group (by default) is that base dimension tables are typically much larger than dimension aggregates, so building fact aggregates that include one or more base dimension tables is expensive in terms of time and disk space.

When this dimension role is added to the default aggregate group, the Other Column Sets listbox is populated automatically with each column set that is declared as Default for the base dimension table to which the dimension role joins. You may assign the default flag of a column set to 1, when it makes sense for that set to be included by default in many fact aggregates.

In all other respects, the Default Aggregate Group is identical to any other aggregate group.

This completes the section on setting up a constellation. To set up another constellation, return to *Setting Up a Constellation,* page 75.

Measures and Ticksheets

> *Note:* Measures and ticksheets will not exist until after you have configured your EpiCenter and have specifically created them using the Web Builder program, which is discussed in Chapter 4.

A measure is a single business calculation, and a ticksheet is a user interface form in Epiphany available to end users for constructing queries. Measures and ticksheets are integral components of the Clarity and Relevance applications.

If measures or ticksheets exist for this constellation, right-clicking the plus sign next to a Measures or Ticksheets folder displays their names. Ticksheets have associated dialog boxes. Double-clicking a ticksheet dialog box (Figure 3-14) enables you to view a list of Saved Queries (reports) for a ticksheet and to view the assigned permissions for these queries.

The Saved Queries panel of the dialog box lists all Saved Queries for this ticksheet. You may select a saved query and click Remove to delete it. The entire list can be deleted; for example, you may want to remove "orphaned" queries. These are queries that cannot be accessed by anyone via Epiphany's front-end Application Server, and may occur when a user is deleted.

Select the Only Show Orphans option in the dialog box to display orphaned queries for deletion.

The permissions for the selected saved query are shown in the lower panel of the tab. You may use the Permissions tab to delete (but not add) individual permissions for a saved query. Select the owner and click Remove.

Figure 3-14 Ticksheet Dialog Box

See *Security* on page 109 for a discussion of permissions.

Basic Concept: Uniform Transactional Data

A transaction represents an event in time. All data in an EpiMart fact table is stored in transactional format. For certain business entities, this format makes intuitive sense, such as in the case of an invoice in which a record in EpiMart represents the shipment of a product to a customer at a certain point in time. For the most part, this event cannot be changed; it simply happened and is stored as such in an EpiMart fact table.

Other business entities are not so easily made into transactions. When a customer calls to order a product, he might choose to order 10 units. In the Epiphany system, this fact is entered as a transaction with quantity 10 for Product P1 to Customer C1.

Order Fact

Customer	Product	Date	Quantity
C1	P1	6/1/1998	10

However, if the customer calls back the next day and changes the order to 15 units, then instead of entering a separate transaction of 15, Epiphany's extraction machinery enters the *difference* of 5 as a new transaction on that second day.

Customer	Product	Date	Quantity
C1	P1	6/1/1998	10
C1	P1	6/2/1998	5

Similarly, in the case of Inventory, Epiphany tracks changes in inventory as transactions instead of restating the reported inventory from the outside world. For example, if Customer C1 *reports* inventory for Product P1 as:

Reported Inventory in Source System

Date	Quantity On Hand
6/1/1998	10
6/8/1998	12
6/15/1998	5

In Epiphany's fact table the same information is represented transactionally as:

Inventory Fact

Customer	Product	Date	Change In Inventory
C1	P1	6/1/1998	10
C1	P1	6/8/1998	2
C1	P1	6/15/1998	-7

Of course, an end user who asks for a report of inventory by week wants to see the data reported in Clarity as it is reported by the source system. To accomplish this, use Web Builder to define a measure that has a backlog type that makes this measure an accumulator over time. This causes the transactions to be reassembled back into the reported format.

Why go to the trouble of disassembling orders and inventories into transactions, only to reassemble the previous format at output time? The reason is that transactions are the most additive way to store data, which means that transactions can be recombined along arbitrary dimensional boundaries.

The way in which Inventory data is reported above can easily answer queries of the form "Inventory by *week*." This data, however, cannot easily be used to answer queries by month or year without some external knowledge of which week is the end of the month. Today's RDBMS engines cannot efficiently handle these types of queries without Epiphany's transactional storage. Note that the inventory for the month of June can be calculated by adding up all the transactions that occur between 6/1 and 6/30, yielding an ending inventory of $10 + 2 - 7 = 5$.

> *Important.* All source system data must be made into transactions. Keep this in mind when you author an Epiphany extractor or choose a semantic type.

Extraction

After you have configured your EpiCenter, you may use the Extraction folder to set up the extraction process described in Chapter 2. The Extraction folder consists of the sub-folders Data Stores, External Tables, Extractors, and Jobs, each of which is discussed below.

Data Stores

Use the EpiCenter Manager's Data Store dialog box to create as many data stores as are appropriate for your site. Each extractor has a single input and output data store. Because jobs consist of multiple extractors, a job has multiple input and output data stores. For example, one job might extract data from two source systems into one EpiMart, and another job might extract data from a source system into external tables and from external tables into staging tables. In the second job, the external tables serve as both input and output data stores to different extractors.

The Data Store Dialog Box

Each data store has a Data Store dialog box (see Figure 3-15), and this dialog box has two tabs: General and Properties.

The General tab has two panels: Data Store Type and Data Flow. The Data Store Type panel of the General tab is where you enter the name and description for the data store and select its type: Microsoft SQLServer, Oracle, Generic ODBC Data Source, or File.

The Data Flow panel of the General tab has these options:

- *Allow Use as Input Data Store* allows the data store selected in the Extractor dialog box *(Figure 3-17, page 95)* to be used as the input of an extractor.

  The source systems listed in the Source System drop-down list serve to distinguish source system identifier keys that may clash between different database systems. For example, if an account has an ERP and an

SFA system, each of which has a Customer table, both may contain a customer number 100. To distinguish between these two records, two different source system identifiers must be used.

Important: Fact rows will join only with dimension rows that have the same source system identifier. For this reason, unless the site actually uses two or more logical source systems, select Datamart Source from the Source System drop-down list (the default).
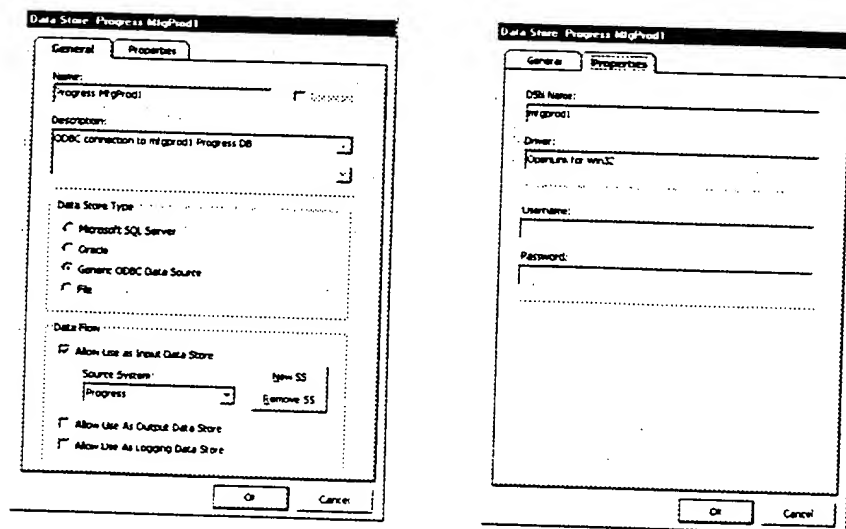
Figure 3-15 Data Store Dialog Box: General and Properties Tabs

- *Allow Use as Output Data Store* allows the data store selected in the Extractor dialog box *(Figure 3-17, page 95)* to be used as the output of an extractor.

  De-selecting this option ensures that one does not accidentally write data to a database that is read-only; the data store name does not appear in the list of available output stores for extractors.

- *Allow Use as Logging Data Store*

  Epiphany has special metadata tables for the purpose of logging its activity. Normally, this logging is written to the EpiMeta database.

  Logging to the EpiMeta, however, may increase its size significantly, so you may prefer to log to another data store. You can use the *logging_mssql.sql* script (included with the Epiphany software) to create a new logging database. If you run this script on another database, then a new data store can be used to log extractions.

  Note: The *Allow Use as Logging Data Store* option must be selected in the Data Store dialog box. The purpose of this option is to ensure that logging is directed to the proper logging data store.

The Properties tab has different fields, depending on the data store type. Use the Properties tab to enter these fields:

- For SQLServer, enter the server's name, database name, username, and server's password and version.

- For Oracle, enter the SQL Net name, username, the server's password, and the version number (Oracle 7 or Oracle 8).

- For a generic ODBC data store, enter the DSN name and ODBC driver name, and server administrator's username and password. (You must also set up a DSN for the data store system using the Data Source Administrator in the Windows NT Control Panel.)

- For the *JobFileLog*, enter its file directory path.

Modifying the Default Data Stores

You will need to modify the default data stores, which are *EpiMart. JobFileLog.* and *LoggingDB.* Double-click their folders to open them and fill out the dialog boxes as follows:

- *EpiMart* specifies your EpiMart as a data store.

  The only value that you can configure is the name of your EpiMart. Use the Properties tab of the EpiMart Data Store dialog box to enter this database name, for example *Corp_EpiMart.*

- *JobFileLog* specifies the data store for EpiChannel job logs.

  In the General tab, the settings are correct for the default usage. The Data Source Type is a file, and the Data Flow is set to *Allow Use as Logging Data Store.*

  Click the Properties tab, and change the directory for the *JobFileLog* from CHANGE ON INSTALL to the correct directory name. A valid directory name is required for a successful extraction. Leave the file name blank.

- *LoggingDB* specifies the data store for the EpiChannel logs.

  In the General tab, the Data Source Type is your server, and Data Flow specifies: *Allow Use as Input Data Store* and *Allow Use as Logging Data Store.*

  You can accept the default values for the log database's file name and directory path, unless you have located the log in a different database.

  Note: $$DEFAULT refers to the current EpiMeta.

External Tables

Extraction statements are sometimes directed to load external tables that serve as intermediary tables for multi-staged extraction. Epiphany supplies one external table called *last_extract_date*. Use of this table is recommended, but optional. See *External Tables*, page 24.

To define external tables and their columns:

1. Choose New External Table from the Extraction menu.

2. Enter the name and any description for the table in the External Tables dialog box (see Figure 3-16).

3. By default, the data in the external tables is deleted (truncated) after extraction. In most cases, you will accept this default.
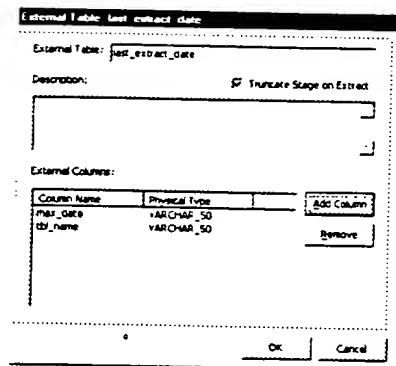
Figure 3-16 External Tables Dialog Box

To add a column:

1. Click Add Column and enter the name in the External Column dialog box.

2. Select the physical type. See Appendix E for descriptions of these physical types.

Add as many columns as necessary. After you click OK in this dialog box, the columns are added to the Column Name list in the External Tables dialog box.

Extractors

In general, SQL statements extract data and semantic instances merge data. The extractor SQL statements may be either stand-alone or extraction (pull/push) SQL statements. Stand-alone statements are typically used to produce a side-effect, such as creating or dropping indexes.

To define an extractor:

1. Choose New Extractor from the Extraction menu.

2. Enter the extractor's name and description in the Extractor dialog box (see Figure 3-17).

3. Select the appropriate input and output data stores from the drop-down lists. *EpiMart* and all data stores that are enabled for input/output appear in these lists.
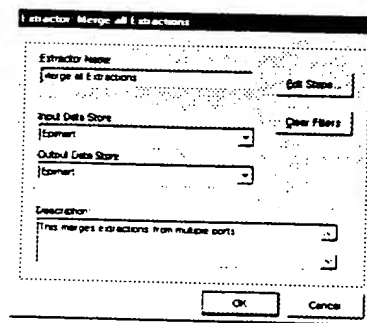
Figure 3-17 Extractor Dialog Box

4. Click the Clear Filters button to delete previous extractors from the system's memory. The effect is the same as if you were starting over, and all data will be retrieved. (See *Extracting New Rows Only*, page 41 for more information.)

5. Click the Edit Steps button to open the Extractor Steps dialog box (see Figure 3-18). The buttons on the right side of this dialog box are the means by which you define a new group, new SQL, or a new semantic instance.

The Extractor Steps Dialog Box

Figure 3-18, page 97 shows an Extractor Steps dialog box in which extractor steps have already been defined. All of the EpiCenter's extraction groups, which consist of extraction steps) appear in the right side of the dialog box.

When working with the Extractor Steps dialog box, keep these definitions in mind:

- A *job* consists of extractors, which can be shared between jobs. (A job also consists of system calls.)

- An *extractor* consists of extraction groups, which can be shared between extractors.

- An *extraction group* consists of extraction steps that are either SQL statements or semantic instances.

When extraction groups have been defined, you may select the ones for this extractor and move them to the Extractor Steps listbox on the left. Use the arrows to move them over, and the Up and Down buttons to order them in the list.

Because an extractor step must be enabled (checked) to be used in a job, de-select any extractor step that you do not want to be run. You may de-select an extractor step to restart a job that failed, or to debug a subset of the steps. In most cases, if a job fails, you should re-run the entire job.

New Group/
SQL/
Semantic
Instance

Extraction
Groups

Extraction
Steps (red)
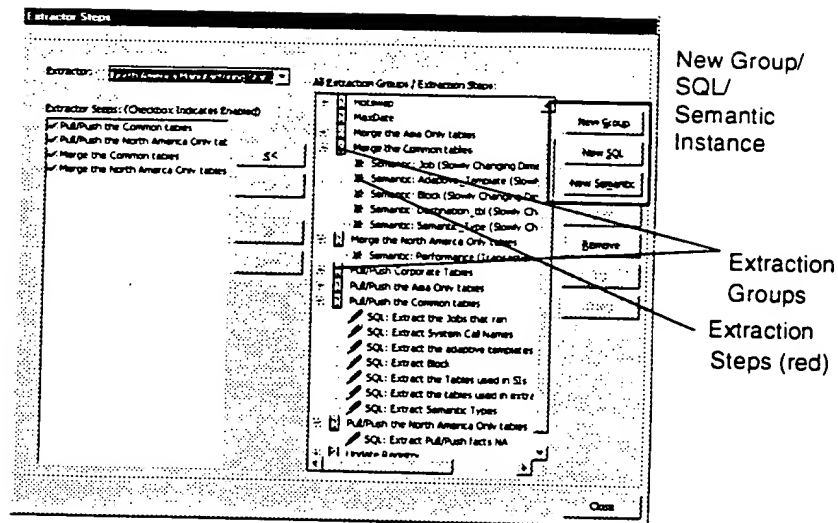
Figure 3-18 Extractor Steps Dialog Box

Notes:

The End of Extraction extractor is provided by default. See *External Tables*, page 24 for a description.

You may right-click a group or an extraction step to display a contextual pop-up menu (for adding a New Group, New SQL, and so forth).

Extraction Group

To define an extraction group:

1. Using the Extractor Steps dialog box, click New Group.

2. Enter the group name in the dialog box; for example, *Order Raw* to extract raw data from the Order master table on the source system.

   This name appears in the All Extraction Groups/Extraction Steps list in alphabetical order.

Extraction Steps for the Group

Next, you need to define the extraction steps for the group. This group can contain any number or combination of SQL statements or semantic instances. If the step requires SQL. EpiCenter Manager provides a template for sample SQL.

To define SQL for one of the steps in this group:

1. Select its group name in the All Extraction Groups/Extraction Steps listbox, and click New SQL.

   The SQL Statement dialog box is displayed (see Figure 3-19). It has three tabs, General, Description, and Store Types. (Use the description tab to add a description for your reference.)
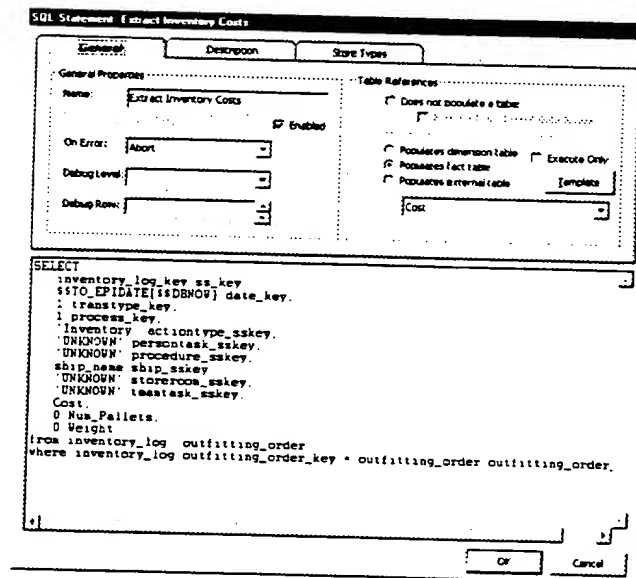
Figure 3-19 SQL Statement Dialog Box

2. In the General Properties panel of the dialog box. enter the step's name (as it will appear in the All Extraction Groups/Extraction Steps listbox). For example, a step named *Customer Raw* extracts raw customer data. The step's name must be unique.

3. You can define a step but not have it execute by de-selecting Enabled.

4. Select the action to be taken if there is an error with the step. The default is to abort the step. (You can also ignore the error, or have it print to a log.)

5. Select the debug level. These levels correspond to the verbosity levels on the **extract** command line. See *EpiChannel Debugging Levels*, page 50.

6. Select the row number at which the debug level you selected above should go into effect, if applicable. If you leave this field blank, the debug level will take effect immediately.

7. In the Table References panel of the SQL Statement dialog box, select whether the step:

    - does *not* populate a table (and if this is true, select whether it executes against input data store, or retains its default behavior of executing against the destination data store);

        —or—

    - populates one of the following types of tables: a dimension table, a fact table, or an external table.

    Most SQL statements are used to populate staging tables (or sometimes, external tables). A SQL statement may, however, be used to achieve a side-effect, in which case you will set it to "not populate a table." In this case, the statement is executed, and any returned results discarded.

8. If the step references a table, select the table from the drop-down list.

9. Execute Only. *(This option is not supported for this release.)*

    Select this when SQL is to be expanded to reflect the structure of one of these tables, but the returned rows from the SQL statement are not meaningful. Epichannel will not insert rows in the statement.

Extraction SQL is executed against the input data store and must be in the dialect of that database engine. The results are stored in the destination data

store. EpiChannel automatically resolves SQL dialect problems if you use the Epiphany database-vendor independent macros consistently in your SQL. (See Appendix B, *Epiphany Macros*, for more information.)

Display Sample SQL

If the step populates a table, clicking the Template button displays sample SQL for the step in the lower half of the dialog box. The SQL will need to be modified to contain the FROM and WHERE clauses appropriate for the tables in the source system, but the template lists the columns that must be returned. It does not matter in what order the columns are returned as long as they have the proper column name (and "extra" unused columns may be returned).

Restrict Data Stores for the Extraction

To restrict the input/output data stores for this extraction:

1. Specify them in the Store Types tab of the SQL Statement dialog box.

2. Click Add Type to add a type.

   If a type is listed, the statement is disabled for any other type except those listed. For example, you could use the same SQL twice, once with Microsoft SQLServer syntax and once with Oracle syntax, and have one or the other statements automatically disable itself when the source database is of the wrong type. This is of value when the group that contains the SQL statement is shared by multiple extractors. Another approach to handling SQL dialect problems is to use Epiphany SQL replacement macros instead of database vendor-specific constructs. See Appendix B, *Epiphany Macros*, for more information.

Additional Steps

To create additional steps within the same group, click OK to return to the Extractor Steps dialog box, and repeat the appropriate steps as described above.

Semantic Instance

To set up a semantic instance:

1. Either create a new group or select an existing group in the Extractor Steps dialog box.

2. Click the New Semantic button. The Semantic Instance dialog box is displayed.

3. Select the fact or dimension table that the semantic instance references.

4. Select the Object from the drop-down list. An object is either a fact table or a base dimension table (the one to be operated on by the semantic type).
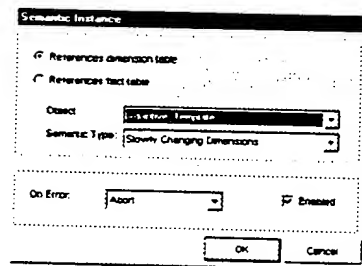
Figure 3-20 Semantic Instance Dialog Box

5. Select the semantic type from the drop-down list. See Appendix G for descriptions of semantic types.

6. Select the action to be taken upon error: abort, ignore, or print. Enabled must be selected for this action to take place.

7. To create additional semantic instances for this group, repeat Steps 2 through 4.

Jobs

Use the Job dialog box to order the job steps (extractors and system calls) that comprise a job. You may use the Add Job Steps dialog box (available through the Job dialog box) to create system calls.

An EpiCenter has a default job consisting of three default job steps: the Aggbuilder system call for aggregation, the End of Extraction extractor, and the Refresh system call for refreshing the Application Server. (For the Refresh system call to work, the Application Server must be configured as described in Appendix A.) You may customize the default job and create as many other jobs as necessary.

To open the default Job dialog box, double-click it. To open a new job dialog box, right-click the Jobs folder and select New Job.

The Job dialog box has two tabs: General and Job Steps (Figure 3-21).
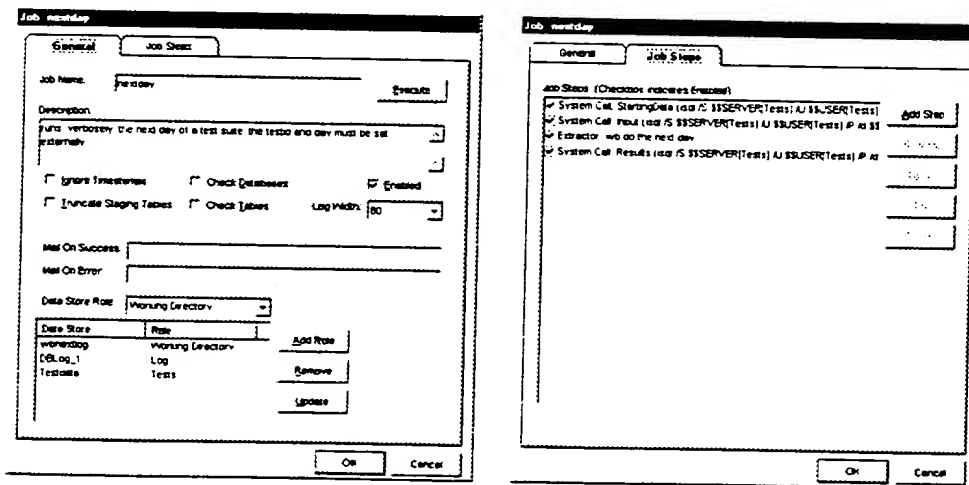
Figure 3-21 Job Dialog Box

The General tab provides options you can set that control the following aspects of a job:

- For a job other than the default one, assign a name and description.

- Enable a job. EpiChannel executes enabled jobs only.

  New jobs are initially disabled because their setup is not complete. You may, however, disable a completely functional job in some circumstances to accommodate system changes. For example, if a database is in the process of being moved or repaired, jobs that populate that database could be disabled as a protection against accidental execution.

- Set up the data stores for the job (by default *EpiMart* and *JobLog*) and indicate their roles; for example, *working directory* or *log*. See *EpiChannel Output*, page 52.

- Select the width of the log, either 80 or 132 columns. Select 80 for VGA monitors and 132 for SVGA monitors.

- Assign addresses for e-mail notification of the job's success or failure. See *Configuring E-Mail*, page 106.

- Request that before executing the job, EpiChannel check that all databases referenced by any job databases and all EpiMart tables are available. Unless you are sure that this is true, select this option.

- Request that all timestamps be ignored during job execution (Ignore Timestamp).

  For example, by ignoring timestamps, you could retrieve all rows, without any date filters (based on EpiChannel's special filtering used for incremental extractions).

- Turn off the truncation of staging tables. (Use this only in development mode to restart a job.)

- Execute a single job.

Click the Execute button in the Job dialog box (Figure 3-21) to execute this job. The Execute Job dialog box shown in Figure 3-22 is displayed. Use this dialog box to:

- Run the job as a trial run.

- Enter a new instance name (by clicking the New button).

- Select a debug level that corresponds to the EpiChannel **(extract)** verbosity levels (although you cannot specify row numbers). This debug level takes effect before the first SQL statement.

- Indicate the total number of rows to be transferred in a pull/ push operation: all rows or the first N rows. This value is reset with each extraction statement.

Using the Execute button is equivalent to invoking the **extract** command with this job as the job option. The Execute button, however, also supplies the database name, password, and so forth, so that **extract** does not depend upon the Registry entries.

Note: If you have multiple EpiCenters, be aware that a job you execute via the Execute button in *EpiCenter A* could be run against *EpiCenter B* if you are using EpiCenter Manager to browse EpiCenter B at the time of the run. This is because the extraction run uses the default values from the Registry, and these values apply to the "current" EpiCenter. This error

will not occur if you use the **extract** command when browsing another EpiCenter.
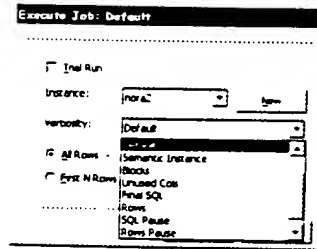
Figure 3-22 Execute Job Dialog Box: Selecting Debug Level

Use the Job Steps tab to do the following:

- Enable job steps by selecting them in the check box to the left of their names. Only enabled job steps are run.

- Change the order of job steps, using the Up, Down, and Remove buttons.

- Update the job definitions in metadata when you modify the job steps. (Click Open to open another dialog box in which you can update a job step.)

Adding Extractors and System Calls as Job Steps

You will also use the Job Steps dialog box to add one of your defined extractors as a job step and to create system calls. Click the Add Step button, which displays the dialog box shown in Figure 3-23.

To add an extractor:

1. Select Extractor.

2. Select the extractor from the drop-down list, and click OK. The extractor now appears in the Job Steps list.

As mentioned, sites with more complex databases may require multi-stages and additional commands, such as lookup tables, aggregation splits, gathering data into ranges (binning), and duplicate detection. For this purpose, you may use system calls, which are executed during a job as if invoked from the console's DOS command line.

To create a system call:

1. Select System Call.

2. Enter the name of the system call, the action to be taken upon error (abort, ignore, or print), the command line, and an optional description.

3. Click OK to add the system call as a step.
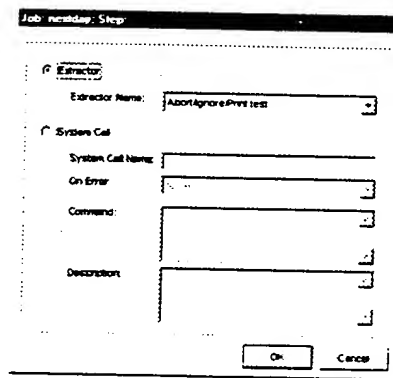
Figure 3-23 Add Job Step Dialog Box

Configuring E-Mail

To have the system automatically send e-mail notification of successful job completion or job failure, you need to set up your e-mail password:

1. Choose Configuration for the EpiCenter menu.

2. In the Configuration tab of the Configuration dialog box, select Mail Password and enter your e-mail password in the value textbox.

3. Click Update.

The Epiphany system uses the machine's default e-mail service. You may enter another mail service, if appropriate.

Truncating Tables

By default, all staging tables and external tables are truncated prior to an extraction. You can set truncation on or off via individual fact, dimension, and external table dialog boxes, or use one dialog box to define all tables in a constellation. They both refer to the same fields in the database.

To define the set of tables for truncation:

1. Choose Edit Truncation from the Extraction menu, or click the Edit Truncation toolbar icon in the toolbar menu.

2. In the Truncate Tables before Extraction dialog box (Figure 3-24), select tables to be truncated.

   All of your tables are listed in columns by type (Dimensions, Facts, and External tables). Check each one in the list that you want truncated. (Clicking the All button selects all of the tables in a column; clicking None de-selects all of them.)
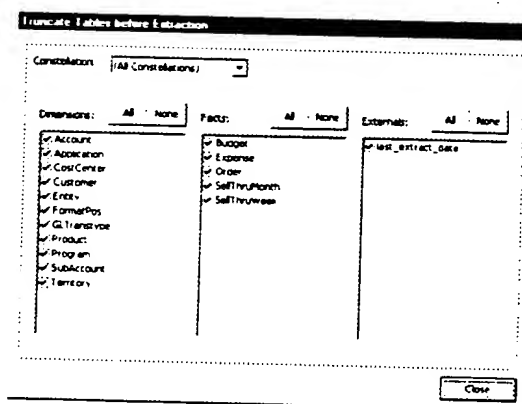
Figure 3-24 Truncate Tables before Extraction Dialog Box

Purging EpiMart Tables

You can remove tables from the EpiMart that are no longer needed. For example, you may have unused tables to delete or, as a result of metadata changes, have decreased the number of aggregates. These higher numbered aggregate tables remain in effect until you purge the EpiMart tables.

You should first try a trial run.

To purge database tables:

1. Choose Purge EpiMart Tables from the EpiCenter menu.

2. Select Trial Run in the Purge tab, and click Go.

After the run, click the Results tab to see which tables will be deleted. If these results are acceptable, return to the Purge tab, de-select Trial Run, and click Go to delete these tables from your EpiMart.
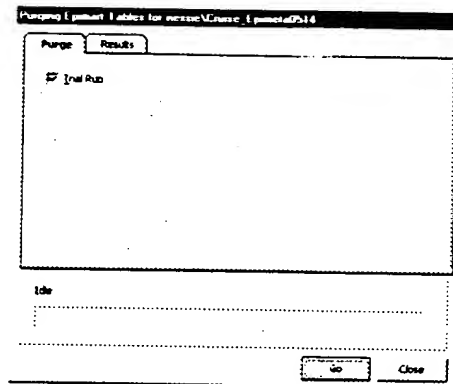
Figure 3-25 Purge EpiMart Tables Dialog Box

Security

The Epiphany system has two separate areas of security:

- *Authentication*, or the ability to log on to the system, which is determined by the Windows NT operating system.

- *Access Rights*, or what the user is allowed to access after having logged on. The Epiphany system can also use Windows NT Groups to administer access rights.

You can set access rights (or permissions) to ticksheets and Saved Queries (pre-built reports). Epiphany's metadata supports these permissions:

- The ability to use a ticksheet.

- The ability to save queries for those ticksheets.

- The ability to access data based on the values of dimensional attributes; that is, to filter database values (for example, to allow some users to see only data for the Western Sales Region).

Before Epiphany users can open any ticksheet, they must be granted access to it. Users have the ability to see and use all ticksheets to which they have access, and to all ticksheets granted access to their groups.

> *Note*: As an administrator, you must use the security dialog boxes in EpiCenter Manager to grant access to a newly created ticksheet before any user can access it.

Users and groups also have access to pre-built reports (Saved Queries) that use the ticksheet. A saved query is simply a list of settings for the various controls that make up the Web page that is a ticksheet. (Epiphany permits global access to certain objects, such as Saved Queries, which are shared objects.) Saved queries can be granted to individual users, to individual groups (and therefore all users in the group), or to all system users (global queries).

There is the concept of a default report for each ticksheet at the user, group, and global levels. A default report is the set of ticksheet settings that a user sees when he or she first opens that ticksheet. In general, the most specific default query is used. Thus if a user-level default query exists, that query is opened in the user's Web browser. If no user query exists but a group default query exists for the ticksheet, the group default query is displayed. If neither of these exists, the global default query is displayed.

You can assign both users and groups explicit permissions for global Saved Queries (see Table 3-2).

Table 3-2 Global Query Permissions

Type	Definition
None	Cannot save global queries.
Save/No Default	Can save global queries, but cannot change which one is the default.
Save and Default	Can save global queries and can also specify which is the default (this is an administrator's role).
Inherit	(For users only) The user's ability to save global queries is determined by the groups to which he or she is a member.

If a user is explicitly given one of the settings None, Save/No Default, or Save and Default, then this determines his or her ability to save global queries. The default setting for users, however, is Inherit, which means that a user is given the access rights that are the least restrictive of the groups to which the user belongs. For example, if the user is a member of a group that has Save and Default access (the least restrictive), then this user also has that access.

You can use EpiCenter Manager to browse the Saved Queries for a ticksheet, as well as view who has access to those queries. See *Measures and Ticksheets*, page 86.

Important: Whenever a ticksheet is created via Web Builder, the administrator should save one global query and make it the default. This allows check boxes to be set to reasonable values for all users when they first use the ticksheet. Because the default global default query serves a special purpose, it does not show up in Epiphany's front-end Report Gallery, which normally shows all Saved Queries to which a user has access.

Dimension Column Access

When a user clicks the Create Report button in a ticksheet to generate a report, some results may be filtered automatically so that the report's output is restricted. Both the User and Group Definition dialog boxes have an Access Rights button, which displays the Access Rights dialog box (see Figure 3-26). Use this dialog box to restrict dimension column access. Select which dimension column to restrict, and which values to *allow* the user or group to see. The values entered in this list should correspond to actual database values in the base dimension table to which that dimension column corresponds.

For instance, selecting *Date.fy_name* as the field with the values 1997 and 1998 causes all reports to be filtered with these values.

If a user is a member of one or more groups, then all group-level restrictions apply to the user. The permissions are additive, so if Group1 has access to Fiscal years 1997, 1998, and Group 2 has access to 1996 and 1997, and a user who belongs to both groups is also explicitly granted access to 1995, then when this user runs a report, values for the years 1995 through 1998 are displayed.

A ticksheet is tied to a particular constellation, and all queries for that ticksheet live within that constellation. When a user runs a report, only those dimension column restrictions that apply to that constellation are used. For instance, if an EpiCenter has two constellations, one for Sales and one for Expense, then it does not make sense to filter on Expense dimensions when the user is working with the Sales ticksheet.
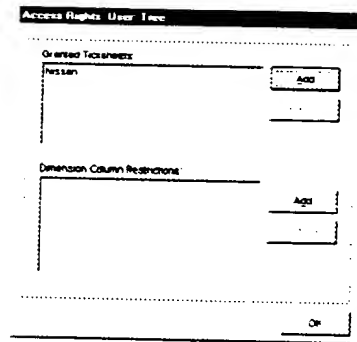
Figure 3-26 Access Rights Dialog Box

Defining Groups

In general, you should attach permissions to groups instead of users to simplify the maintenance of access rights. Users inherit all permissions from all groups to which they belong, except that a user can be denied permission to save global queries (otherwise available to members of his or her group).

You can assign a user's group membership using either the User Definition or Group Definition dialog box. Group-level queries are visible only to members of the group.

Global Queries

Global Queries are saved reports (created via ticksheets) that every user of the system can see and use. Normally, control is set by groups.

The options are—

- Inherit. Select Inherit if you want the user to inherit the same rights to reports as his or her group. This is the default value, and is available only for users, not for groups.

- None. This user cannot save group queries.

- Save/No Default. The user can save group queries, but cannot change which query is the default.

- Save and Default. The user can save group queries, as well as to specify which is the default (this is one of the tasks of a group administrator).

  Note: The user's ability to save group-level queries is independent from his or her ability to save Global queries.

Synchronized Groups

For authentication purposes, new users may be added to the Epiphany metadata tables in an "autopilot" fashion via synchronized groups. You need to precede Windows NT synchronized group names with their Windows NT domain name prefix (and matching name).

When the synchronize option is set, each time a user logs in, the Windows NT security API is accessed for a list of group names to which that user belongs. Any name matching an Epiphany group name causes the appropriate group membership to be affected. If necessary (for example, the first time a user logs in), the user record itself is created. This last feature allows Windows NT administrators to create new Epiphany users.

To define groups:

1. Select New Group from the Security menu. The Group Definition dialog box is displayed (see Figure 3-27).

2. Enter the group name.

3. In the Save Global Type drop-down list, select an option as described in *Global Queries*, page 112.

4. Click Access Rights to grant group members permission for all or a subset of the available ticksheets. Select a ticksheet from the Granted ticksheet list, and click Add to give the group access to that ticksheet.
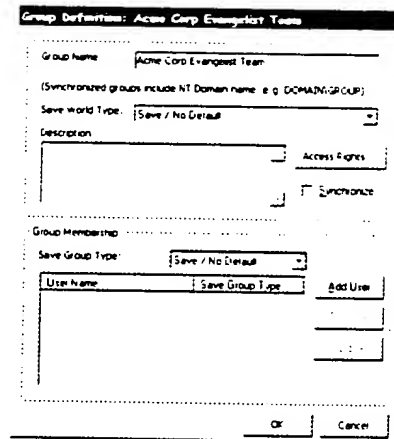
Figure 3-27 Group Definition Dialog Box ·

5. To restrict access rights for the ticksheet to certain rows (row-level security), click the Add button in the Dimension Column Restriction dialog box .
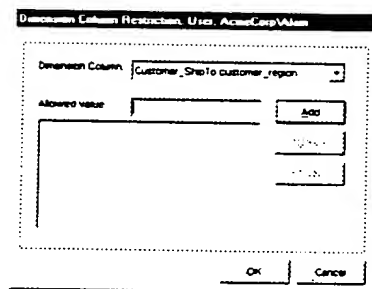
Figure 3-28 Dimension Column Restriction Dialog Box

6. Select the dimension column from the drop-down list. The dimension columns fields display in the listbox.

7. Selecting a name from the listbox displays it in the Allowed Values text box. The Add button specifies the values that a user or group is allowed to see. Select additional fields as appropriate, and click Add to add them to the list.

8. Click OK to return to the Group Definition dialog box.

9. In the Group Membership panel, select the Save Group Type from the drop-down list. The options are—

   - None. Group members cannot save their ticksheet configurations.

   - Save/No Default. Group members can save ticksheets they create, but not default queries.

   - Save and Default. Group members can save ticksheets and default queries.

10. Add users to the group. Click Add User and select group members from a list of defined users.

11. Select the Synchronize option to make group membership for the user a subset of the Windows NT memberships for that user. Synchronization is performed only on groups for which Synchronize is selected in this dialog box.

Defining Users

When new users are entered into the system, they have these default permissions:

- no access to any ticksheets.

- no group membership.

- cannot save any queries.

- can view all data in all dimensions—there are no dimensional restrictions.

To add or modify a user definition:

1. Right-click the Security folder and select New User. The User dialog box (Figure 3-29) is displayed.

2. Enter the person's username and first and last names.

   The first and last names are used for display purposes only. If they are missing, then the username is displayed.

   For Windows NT authentication, enter users with their Windows NT domain prefix to distinguish among users with the same name but who are in different domains.

3. In the Save Global Type drop-down list, select an option as described in *Global Queries*, page 112.

4. If ticksheets have been created, in special cases you may click Access Rights to grant that user permission to view all or a subset of the available ticksheets and ticksheet attributes. Usually, this kind of access is granted for Groups.

Figure 3-29 User Dialog Box

5. In the Access Rights dialog box, select a ticksheet from the Granted ticksheet list and click Add to give the user access to that ticksheet.

6. To restrict access rights for the ticksheet to certain rows (row-level security), click the Add button in the Dimension Column Restriction dialog box.
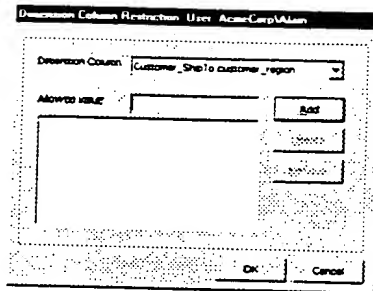
Figure 3-30 Dimension Column Restriction Dialog Box

7. Select the dimension column from the drop-down list.

8. In the Allowed Value text box, enter the name of the field whose values this user will be able to access, and click Add. Enter additional fields as appropriate, and click Add to add them to the list.

9. Click OK to return to the User dialog box.

Basic Concept: Adaptive Architecture

In traditional client/server application environments, changes to an underlying table's schema adversely affect programs that operate on that schema. The goal of Epiphany's Adaptive Architecture is to allow on-the-fly changes to the EpiMart schema while preserving the form and proper operation of the entire Epiphany Application Suite.

A schema change in EpiMart can affect these components:

- Extraction statements that populate these staging tables.

- Semantic instance programs that merge staging data with EpiMart data.

- Aggregates defined on these tables.

- Measures that refer to these tables.

- Ticksheets that refer to columns in these tables.

- Security restrictions on columns in these tables.

The use of a single metadata repository (EpiMeta) ensures that all components of the system receive notice of a change simultaneously; for example, Aggbuilder will not try to build aggregates on tables or columns that no longer exist.

During extraction, semantic Instances are SQL programs that accomplish business transformations on extracted data. Since these programs must be changed in response to schema changes, these programs are compiled on-the-fly at execution. Thus the program uses the latest schema metadata in its construction, which results in a well-formed set of SQL statements.

EpiCenter's Generate Schema command creates and modifies the EpiMart tables. The first time this operation is executed, all tables are built using CREATE TABLE statements. These tables include base dimension, fact, and external tables. Certain metadata fields are used in the actual construction of these tables.

EpiCenter Manager ensures that these fields follow the proper naming conventions for table and column names. These naming conventions are as follows:

- Base dimensions

  The name of the table is taken from the name of the base dimension. Each dimension column becomes a physical column in the EpiMart table with the same name.

- Fact tables

  The name of the table is taken from the name of the fact table. Each metadata fact column becomes a numeric column in the EpiMart table with the same name. Each dimension role and degenerate dimension of the constellation to which the fact table belongs become a foreign key column in the fact table.

- External tables

  The name of the table is taken from the name of the external table. Each external column becomes a column in the EpiMart table with the same name.

Each time Generate Schema is executed, EpiCenter Manager records the schema of the newly built tables in a set of metadata tables called the Actual tables. When subsequent changes are made to the schema definition, EpiCenter Manager examines these Actual tables to compute a delta operation.

Important: If Generate Schema is not run and the schema definition has been altered, neither Aggbuilder nor the Epiphany Application Server will start. Both will recognize a difference between the metadata schema definition and the contents of the Actual tables.

Table 3-3 describes the schema operations.

Table 3-3 Schema Operations

Operation	Action
Add a new base dimension	The table is created from scratch.
Delete a base dimension table	The table is dropped and all dimension roles that use that base dimension are deleted.
Rename a base dimension table	Treated the same as a deletion of the old table and an addition of the new table.

Table 3-3 Schema Operations

Add a new dimension column to an existing base dimension	The existing table is altered to include that column. All existing rows take the default value for that column.
Delete a dimension column from a base dimension	The column is removed.
Rename a base dimension column	Treated the same as a deletion of the old column and an addition of the new column.
Add a new fact table	The table is created from scratch.
Delete a fact table	The fact table is dropped.
Rename a fact table	Treated the same as the deletion of the old name and an addition of the new table.
Add a new fact column	The table is modified to have the new column. All existing rows get the value 0.
Delete a fact column	The column is removed.
Rename a fact column	Treated the same as a deletion of the old column and an addition of a new one.
Add a dimension role	The column is added to all fact tables in that constellation. The value for all existing rows is set to 1, which is a special value that always points to the UNKNOWN row in the base dimension to which that role refers.
Delete a dimension role	The column is removed from all fact tables in that constellation.
Rename a dimension role	Treated the same as the deletion of the old column and an addition of the new one.

Table 3-3 Schema Operations

Add a new degenerate dimension	The column is added to all fact tables in that constellation. All existing rows get the special value UNKNOWN.
Delete a degenerate dimension	The column is removed from all fact tables in that constellation.
Rename a degenerate dimension	Treated the same as a deletion of the old column and an addition of the new one.
Add a new external table	The table is created from scratch.
Any change to an external table including changes to column	The table is dropped and recreated.

Generating Schema

After using EpiCenter Manager to define metadata, you need to convert these definitions into Actual tables (EpiMart).

To generate schema:

1. Choose Generate Schema from the EpiCenter menu.

2. Select Trial Run to see what the results will be without making any changes.

3. By default, only tables that have changed since the last time you generated schema are rebuilt.

   Select Force Rebuild of all Tables only if you want to rebuild *all* of your tables.

4. Click Go (and watch the progress bar).

5. After the trial run is finished, click the Results tab to view which tables were updated.

6. If the results are acceptable, de-select Trial Run in the Schema tab, and click Go.

After the schema has been generated, the system records the current state of the metadata so that the next time the schema is updated, the current definitions can be compared with the new ones, and the appropriate tables can be created or altered as necessary.
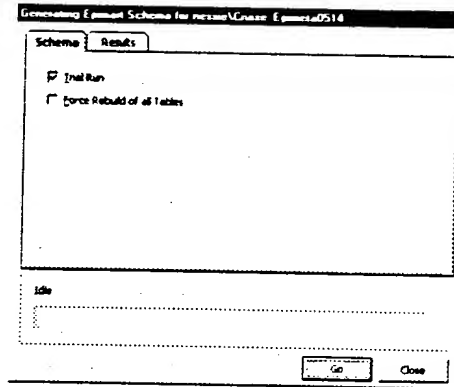
Figure 3-31 Generate Schema Dialog Box

Exporting/Importing Metadata

Epiphany provides a metadata export/import utility for moving metadata between EpiCenters and for backing up the definition of an EpiCenter. To use this tool properly, you need to understand the metadata concepts (see *Metadata Overview*, page 279).

The export operation produces a Microsoft Access database that contains a representation of the metadata that was chosen for export. Upon import, this representation is converted back into valid Epiphany metadata in the target EpiMeta. Using EpiCenter Manager, you have granular control over which

metadata objects get exported during each operation. For example, you can use the same constellation definitions, extractors, and ticksheets in several EpiCenters. Also, during import, you control whether or not to overwrite existing metadata that conflicts with what is in the import file.

Reasons for exporting files other than publishing them for import are to save files for document control (source safe) and to send files as an e-mail attachment; for example, you might e-mail an exported file to Technical Support for analysis.

You can produce a new EpiCenter by importing all or a subset of the following metadata files from an existing EpiCenter: Base Dimensions, Constellations, Measures and Ticksheets created via Web Builder, Data Stores, External tables, Extractors, SQL Templates, Jobs, Groups, and Users.

To import metadata files:

1. Right-click the EpiCenter icon and select either Import Metadata or Import Templates from the New EpiCenter menu.

   An option in the import dialog box allows new data to replace existing entries of the same name. Select Save to overwrite an existing entry of the same name.

2. Select the file name from the *Choose the Name of an Import file of type* dialog box and click Open. Save the file.

See Appendix H, *Export/Import of Metadata*, for more information.

CHAPTER FOUR

Web Builder and Ticksheets

This chapter introduces the Clarity and Relevance ticksheets and explains how to use Web Builder to create these ticksheets. A ticksheet provides users with point-and-click access via a Web browser to integrated information from sources such as sales, finance, and customer support databases. (The term *ticksheet* refers to the fact that users tick, or check, items on a Web page.) Those who use Clarity or Relevance to access their data warehouse do not need to know anything about the underlying data's structure. All they need to know is what kind of information they want to analyze.

Generating a Report

Clarity and Relevance ticksheets are easy to use. (For examples of ticksheets, see Figure 4-1, page 127 and Figure 4-2, page 128.) In general, you will follow these steps:

1. Select the attributes and measurement criteria for a report.

2. Select the display options.

3. Apply filters to refine your search.

4. Click the Create Report button.

A report, or results page, displays in the browser window. If you have questions while using Clarity or Relevance, click the Help button in the left frame of the window for online help.

A description of this procedure follows.

Attributes and Measures

In Clarity, the attributes are arranged by columns and rows. For example, if you select products for the column and years for the row, the report will have columns for each of the designated products in the database and rows for each of the designated years. In Relevance ticksheets, you may select attributes in order to forecast trends, or to analyze highs and lows, best and worst cases, or graphs for various aspects of your business.

A ticksheet has one or more *selection columns* that consist of *selection column elements* (see Figure 4-1, page 127). You may select one element from each column. This combination comprises a specific *measure*, or business calculation. For example, assume you selected Units, Booked, and Net Price. The system recognizes this combination of elements as the measure *UnitsBookedNetPrice* and applies the calculations defined for this measure to aggregate the number of units booked times the net price for each product. The results are placed in each cell, arranged by product and by year. The numbers are totaled (summed) for columns and rows.

To find out what a selection column element means, click its link (see Figure 4-1, page 127), which brings up the Dictionary page where the selection column elements for an EpiCenter are defined. (Each site creates and maintains its own dictionary entries.)
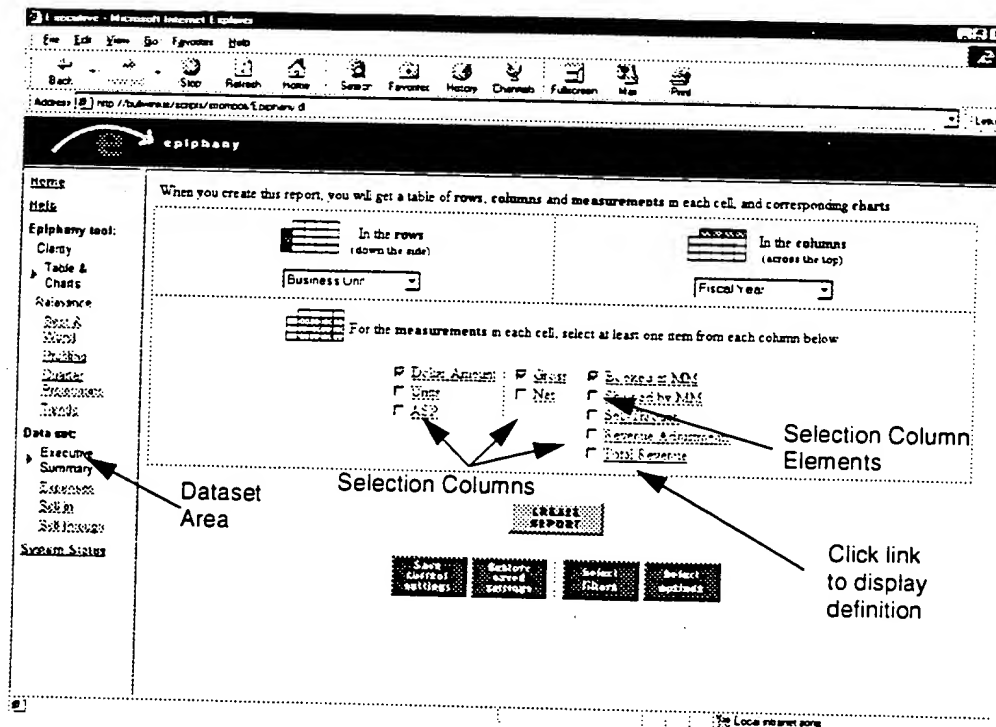
Figure 4-1 Sample Clarity Web Interface (Ticksheet)

Display Options

You can set ticksheet options that enable you to:

- select how numbers are represented.

- receive graphical charts with your report.

- limit the rows that display.

- sort rows either by value or row name.

- display data in either HTML or spreadsheet format.

Click the Select Options button on the ticksheet to open the Options window.

Filters

Rather than receiving a report with the results for all of the criteria you selected, you may limit the data by using filters. For example, you may restrict the search to the last three years, or to products sold only through direct channels.

Click the Select Filters button on the ticksheet to open the Filters window.

Figure 4-2 Sample Relevance Ticksheet

Web Builder Overview

The front-end ticksheets for Clarity and Relevance are created via Web Builder. Figure 4-3 shows Web Builder's system hierarchy. Each Web Builder *ticksheet* is specific to a constellation within an existing EpiCenter. Ticksheets are organized by *datasets*, which are a logical user-interface grouping of ticksheets that display similar views of data. The ticksheet's major components are attributes, selection columns, and filters.

Figure 4-3 Web Builder System Overview

A *measure* is a single business calculation that is usually the result of an arithmetic combination of measure terms. A *measure term* is one component of an arithmetic expression that makes up a measure. It refers to the aggregation of a single fact column, such as *SUM(Order.net_price)* with a particular transaction type. It can be combined with other measure terms to create a composite measure. These calculations are the values returned in the cells of Clarity and Relevance reports.

Measure mapping is the association of a set of selection column elements on a ticksheet map with a specific measure. When the ticksheet user selects elements in selection columns, the system maps this combination to a measure. It then applies this measure's calculations and returns the results of the query.

Measures, as are ticksheets, are constellation wide. The dictionary entries that define measure terms to users, however, apply to the entire EpiCenter.

You will use Web Builder to define measures, dictionary entries, and ticksheet components according to the instructions that follow.

Getting Started

After launching Web Builder, the Web Builder Logon dialog box is displayed. To log on:

- Enter your server system administrator password.

- Enter the name of your server, your username, and the EpiCenter to which this ticksheet belongs.

The values you entered in this dialog box, except for your password, are saved so you do not have to enter them again the next time you log on.

The Web Builder Window

When you log on to Web Builder. the Web Builder window contains the main menu. Descriptions of the main menu. pop-up menus, and Web Builder icons follow.

The Main Menu

You will use the main menu commands to create or modify ticksheets. The main menu always displays the File, Edit, System, Ticksheet, Window, Help, and About commands. The other menus are contextual. The Attribute, Element, Filter Block, and Filter Group menus display when you are working directly with them.

Use the *File* menu to—

* reload the configuration information from the database.

* save a ticksheet.

* quit the Web Builder program.

* select recently opened ticksheets.

Use the *Edit* menu to cut, copy, paste, or perform a *smart paste*. A smart paste is a convenience feature that applies to measures only. See *Defining Measures* on page 136.

Use the *Ticksheet* menu to—

* create a new ticksheet.

* open an existing ticksheet (Figure 4-4) by double-clicking the ticksheet's name in the list of ticksheets.
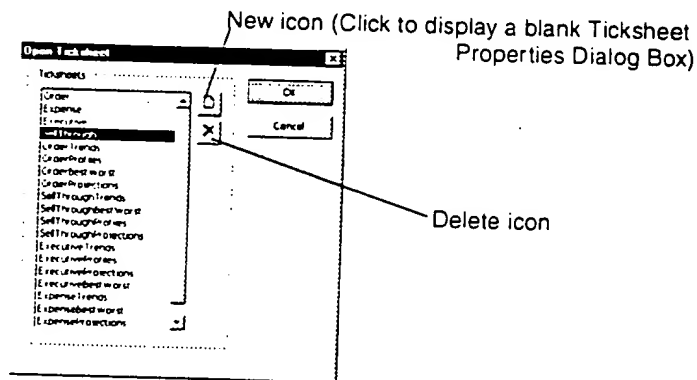
Figure 4-4 Open Ticksheet Dialog Box

- delete a ticksheet. Select it and click the Delete icon.

- open the Ticksheet Properties dialog box for the current ticksheet. You can also click the New icon in the Open Ticksheet dialog box to display this dialog box.

Use the *System* menu to define measures, edit the dictionary, and define datasets and system properties.

The contextual menus display when you are working in the Attributes, Columns, or Filter panel of the Ticksheet dialog box:

- Use the *Attribute* menu to add and delete attributes, and to change attribute properties.

- Use the *Element* menu to add and delete elements, and to change their properties.

- Use the *Filter Block* menu to add, delete, and modify Filter Blocks. Use the *Filter Group* menu to add, delete, and modify Filter Groups. (For definitions of these terms, see *Defining Filters* on page 152.)

Use the *Windows* menu to select an open Web Builder window.

Open the *Help* menu to display online help for Web Builder.

The *About* menu shows the Web Builder version number and statistics for the EpiCenter as of the last time the system was loaded from the database. This includes the number of ticksheets, attributes, measures, dictionary entries, and the total number of objects.

Right-Click for Pop-up Menus

There are pop-up menus for Attribute, Element, Filter Block, and Filter Group. Right-mouse click inside the area where the attributes, elements, or filters are listed in the Ticksheet dialog box to display its pop-up menu. (The area must have items in it for these pop-up menus to display.)
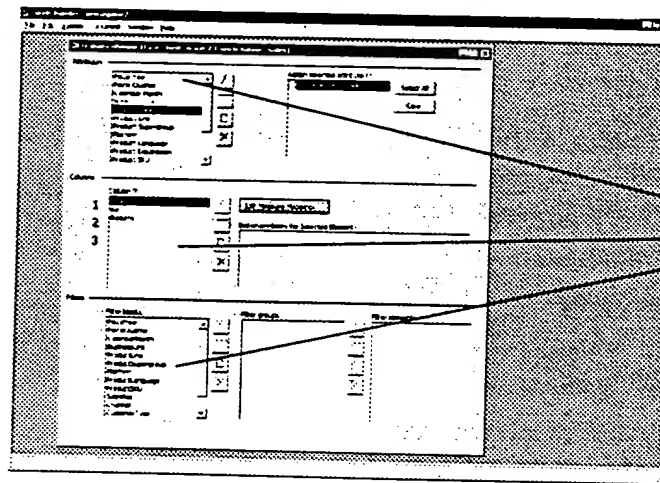
Right-click
to display
pop-up
menu

Web Builder Icons

Use Web Builder icons to—

- move selected items up or down by clicking the directional arrows.

- add new items by clicking the Add a New Item icon.

- delete items by selecting the item and clicking the Delete icon.

Move
Selection
Up

Move
Selection
Down

Add a
New Item

Delete
Selection

Setting System Properties

Before creating a ticksheet, you need to set up system properties. Select Properties from the System menu, which displays the System Properties dialog box (see Figure 4-5).

Enter your company name and the GIF file name for your company logo (see Figure 4-5). This GIF file must be located in the directory named *images* in the Epiphany Web directory. Rename or copy the GIF file to **clientlogo.gif**.

> *Note:* A logo or other mark in the upper-right corner of a ticksheet should be a GIF file that is 225 x 50 pixels. Epiphany recommends that this image be right-justified and placed on a black background. Along each edge, leave a few of the pixels black.

The dataset ordering applies after you have created more than one dataset. Use the up and down arrows to position the datasets as they should appear on the top-level page of your Web interface.
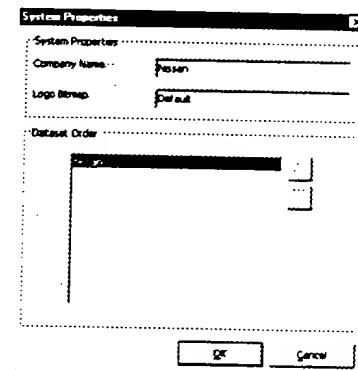
Figure 4-5 System Properties Dialog Box

Defining Datasets

A dataset is a logical grouping of similar ticksheets within a constellation. You can group related Clarity and Relevance ticksheets by dataset. In the browser window, these group names, or labels, appear in the left frame of the ticksheet under the Dataset heading. (See *Sample Clarity Web Interface (Ticksheet)* on page 127.) For example, one dataset may contain Executive ticksheets and another have Summary reports.

Use the Dataset Builder dialog box to set up dataset groupings for a constellation. (Open it by choosing Define Datasets from the System menu.) Use the Ticksheet Properties dialog box (Figure 4-9, page 144) to assign a ticksheet to a dataset.

To add a new dataset category, click the New button and enter the label name. Use the Remove and Rename buttons to modify the listing.
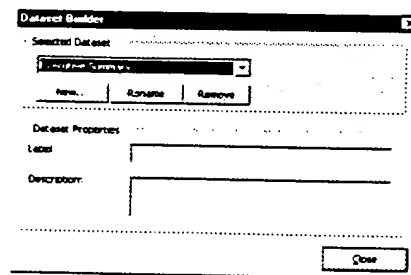
Figure 4-6 Dataset Builder Dialog Box

Defining Measures

The selection column elements that a user checks on a ticksheet will be mapped to a specific measure. You need to inform the Epiphany query machinery of the following:

- which set of selection column elements should map to a specific measure (as described in *Measure Mapping* on page 149). and

- how to convert the measure into the numeric data of the report cells. You will do this by defining measures.
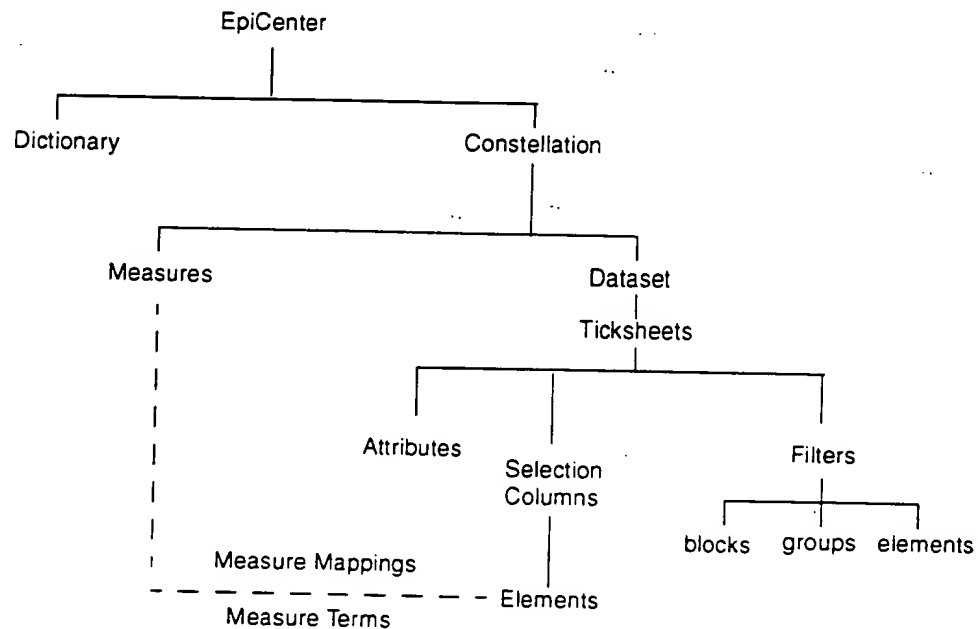
As mentioned, a measure is a single business calculation that is usually the result of an arithmetic combination of measure terms. A measure term is one component of an arithmetic expression that comprises a measure, and it refers to the aggregation of a single fact column with a particular transaction type.

Use the Measure Builder dialog box to define a measure:

1. Select Define Measures from the System menu. The Measure Builder dialog box (Figure 4-7) is displayed.

2. In the Measure Selections panel, click New and enter a name for the new measure. Typically, measure names are a combination of the selection column element names; for example, the measure ASPBookedOrdersGross indicates that the user selected Average Sales Price, Booked Orders, and Gross.

3. Select the unit of measurement from the Units drop-down list.

   You can define these options using in the Measure Units tab of the EpiCenter Manager's Configuration dialog box (see *Measure Units* on page 247).

   Example units of measurement are—

   - Currency (for money units)

   - Percent (for percentages)

   - Units (for the count of an item)
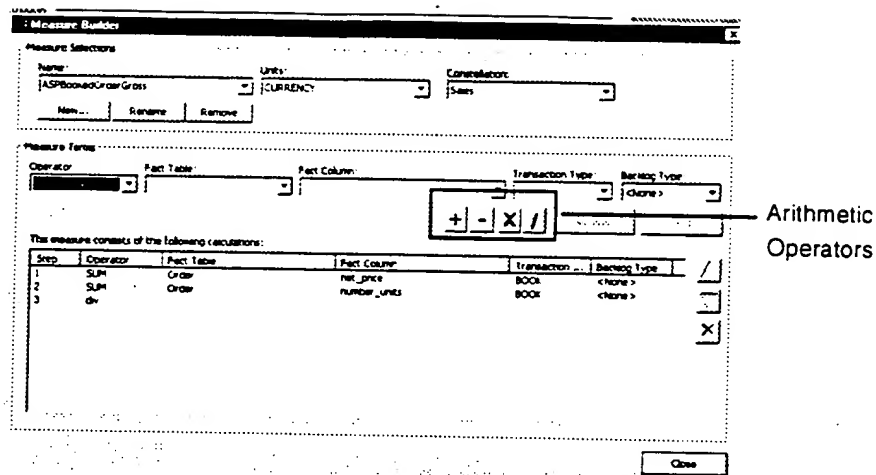
Figure 4-7 Measure Builder Dialog Box

4. Select the constellation to which the measure will belong.

Notes:

Changing the constellation for an existing measure erases all measure terms created for this measure.

Defining Measure Terms

Use the Measure Terms panel of the Measure Builder dialog box to define the steps that determine how the measure is calculated. Each step applies to a specific fact table, fact table column, transaction type, or backlog type (if applicable).

Web Builder converts these steps to appropriate SQL SELECT statements. You will use Reverse Polish Notation to construct measure definitions. See *Reverse Polish Notation* on page 140 for more information.

To define a step for a measure term:

1. Select one of the SQL operators from the drop-down list: SUM, MIN, MAX, AVG, COUNT, COUNT DISTINCT (or the negative values of these operators, such as -SUM or -MAX).

2. Select the fact table where the data resides, the fact column, transaction type, and backlog type.

   The backlog types are BEGIN or END; or use <None> if the backlog type does not apply. Use a backlog type when a measure term (a single line of the measure definition) should exhibit accumulating behavior. Normally, when running a report of Sales by Month, for example, the report shows only the sum of transactions that occur in each month of the report. When a backlog type is used, however, the columns of the report show accumulated values from previous months, in addition to the current month. Use BEGIN to show the accumulated value at the beginning of each period, and END to show the ending accumulated value.

   For example, assume that report of sales by month transactions shows $10 for June, $20 for July, and $40 for August. A report of the beginning backlog shows the accumulated value at the beginning of each month:

June	July	August	September
$0	$10	$30	$70

A report of the ending backlog shows the accumulated value at the end of the month:

June	July	August
$10	$30	$70

Notes:

You can cut, copy, and paste measures using the Edit menu. The Smart Paste command performs a standard paste, and allows you to substitute values in the Fact Table, Transaction Type, or Backlog Type fields. For example, a measure that relates to the Orders fact table can be copied and pasted to apply to a Budget fact table. You can copy and paste a measure defined for the transaction type Book, and then substitute a Ship transaction type.

See Appendix C for more information about transaction types.

3. Click Add to add this as the first step in the lower panel.

4. If appropriate, add another measure term by repeating Steps 1 through 3.

5. Click the appropriate arithmetic operator to be applied to the measure terms: add (+), subtract (-), multiply (x), or divide (/). (See Figure 4-7, page 137 for the location of these operators.) See *Reverse Polish Notation* on page 140 for examples.

To change an existing measure term, select it from the Name drop-down list and modify as described above. Click Update to save the changes, which will be reflected in the dialog box.

Use the Remove button to delete a measure, and the Rename button to rename a measure.

Reverse Polish Notation

The Measure Terms panel in the Measure Builder dialog box uses Reverse Polish Notation[3] (RPN) to construct measure definitions. RPN operations are performed in a last-in, first-out (LIFO) basis. All of the values to be operated upon are placed in a stack. Then the top two are operated upon and the result of that operation is placed in the stack, replacing the previous two values. Then the next top two are operated on and the result placed in the stack, and so forth.

For example, using Reverse Polish Notation for this calculation:

$$1 + (2 * 3) = 1, 2, 3, *, + \text{ in RPN}$$

means that 1, 2, and 3 are placed in the stack. The last two items (3 and 2) are multiplied, and the resulting value 6 is placed in the stack. The stack now holds 6 and 1, which are added, and the result 7 is placed in the stack.

In contrast, applying RPN to the calculation:

$$(1 + 2) * 3 = 1, 2, +, 3, * \text{ in RPN}$$

means that 1 and 2 are placed in the stack. These items are added, and the result 3 replaces them. Next, the value 3 is placed in the stack (the stack now holds 3 and 3). These items are multiplied and the result is 9.

Example 1 below shows how RPN is used to define a measure definition (in the Measure Builder dialog box). The Average Sales Price (ASP) for Booked/Gross Orders equals the total number of dollars received, divided by the total number of units shipped. This is represented as the following SQL SELECT statement:

```
SUM (Order.net_price)
SUM (Order.number_units)
div
```

Example 1

---

[3] Reverse Polish Notation is named for its Polish inventor, Jan Lukasiewicz.

This SELECT statement is calculated using RPN as follows. The sum of all the Order net prices is calculated and placed in the stack. Then the sum of all the number of units is calculated and placed in the stack. Next, the division operator is applied to the top two items in the stack. It takes the next item in the stack, the total net price received, and divides it by the total number of units shipped, and the result equals the ASP.

For the operators that apply to the aggregates, use the arithmetic operators: add, sub, mult, and div.

Example 2 shows a similar calculation. This time the ASP is calculated for booked net orders (booked orders minus returns). Here the sums of two Order net prices are added (the returned items are represented as a negative number) and placed in the stack. Then the sums of two Order number of units (the returned items are represented as a negative number) are added and placed in the stack. The remainder of the calculation is the same as for Example 1 above.

```
SUM (Order.net_price)BOOK
SUM (Order.net_price)BOOK_RETURN
add
SUM (Order.number_units)BOOK
SUM (Order.number_units)BOOK_RETURN
add
div
```

Example 2

Defining Dictionary Entries

Selection column elements in Clarity and Relevance ticksheets, such as Units, Gross, and Sell-Through, are hyperlinked to a dictionary page. When a user clicks a link, a dictionary page displays which defines it. These dictionary entries help users understand your terminology.

Selecting Edit Dictionary from the System menu displays the Dictionary Builder dialog box (see Figure 4-8).

To add a dictionary entry, click the New button and enter the selection column element's name in the Name dialog box, and click OK. Enter the definition in the Entry Text area of the Dictionary Builder dialog box. Click Close to save the entry.

To remove an entry from the Web Builder Dictionary page, select the entry from the Entry Name drop-down list and click the Remove button.
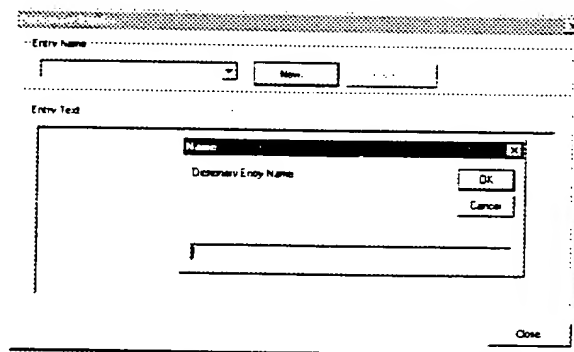
Figure 4-8 Dictionary Builder Dialog Box

You can also use the Column Elements Properties dialog box (Figure 4-12, page 149) to add and modify dictionary entries for a selection column element. To open this dialog box, double-click the selection column element, or select it and choose Element Properties from the Element menu. Click Dictionary Editor to open the Dictionary Builder dialog Box.

Note that a ticksheet selection column element's dictionary entry (if any) displays in the Columns panel of the Ticksheet dialog box when selected.

Creating a New Ticksheet

Important: Creating a ticksheet does not automatically make that ticksheet available to end users. You must first use EpiCenter Manager to grant this permission to one or more Epiphany users and groups.

Each time a ticksheet is created via Web Builder, the administrator should use EpiCenter Manager (see *Global Queries* on page 112) to save one global query and make it the default. This allows check boxes to be set to reasonable values for all users when they first use the ticksheet. Because this global default query has this special purpose, it does not show up in the Report Gallery. (The Report Gallery is a feature of Clarity and Relevance that lists the Saved Queries that a user may access.)

You may start building your ticksheet from scratch as described below. For every ticksheet, you will define attributes, columns, and filters. After you complete a ticksheet and save its configuration, it is available for a front-end user to open in a Web browser.

To create a new ticksheet, choose New from the Ticksheet menu. Use the panels in the Ticksheet Properties dialog box (Figure 4-9, page 144) to define the ticksheet.

- Ticksheet Properties

  Enter the name of the ticksheet (no spaces are allowed), the label name (the name that the user sees in the Web browser), and any descriptive text.

- Display Options

  Select the ticksheet type. See *Ticksheet Types* on page 145 for a description of each.

  Select the number of columns that will appear on the Clarity ticksheet (Figure 4-1, page 127). A column consists of selection column elements.

- Constellation and Dataset

  Select the constellation for this ticksheet from the drop-down list.

  Select the dataset that the ticksheet belongs to. Datasets are subsets of the constellation's ticksheets listed in the left margin of Clarity and Relevance Web pages (see Figure 4-1). See *Defining Datasets* on page 135 for more information.

- Data Copy

  You may copy attributes, columns, or filters from other ticksheets within this constellation. This is a fast way to create a ticksheet when it shares a subset of another ticksheet's properties.

  Note: If you want to copy columns, be sure that the ticksheet you copy columns from has the same number of columns as your new ticksheet.

After you complete the dialog box and press OK, a new ticksheet with these properties is displayed.
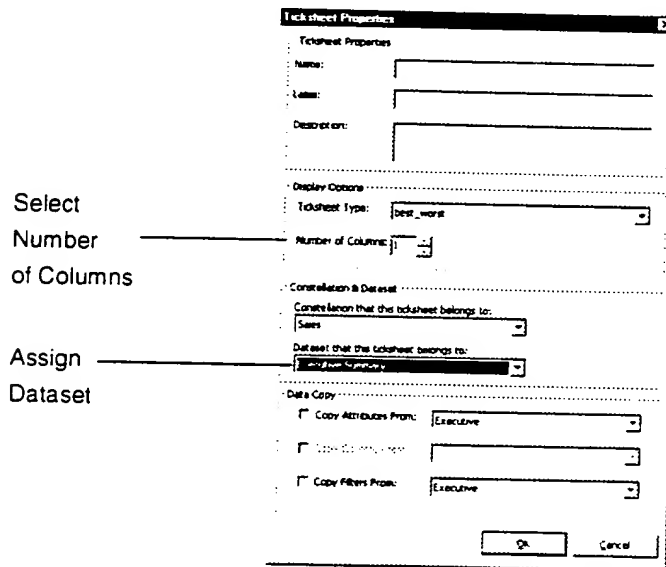
Figure 4-9 Ticksheet Properties Dialog Box

Ticksheet Types

The ticksheet types you may select in the Ticksheet Properties dialog box are:

- Best & Worst

  Relevance Best & Worst searches thousands of Clarity tables automatically and builds a list of the most interesting results within those tables.

- Clarity

  A Clarity ticksheet produces reports for selected attributes (the columns and rows of the report) with measure values inside the cells. Corresponding bar and pie charts are also provided.

- Momentum

  [*Not available in this release.*]

- Profiles

  Relevance Profiling shows a page of thumbnail charts that help you quickly understand your data.

- Quarter Projections

  Relevance Quarter Projections answer the question, "Given where I am today, and given the history of previous quarters, how is this quarter likely to end? Will I meet my goals?"

- Trends

  Relevance Trends fits a smooth curve to time-varying data, and forecasts one or more periods into the future.

The Ticksheet Dialog Box

The ticksheet dialog box (Figure 4-10) has three panels: attributes, columns, and filters. These sections correspond to the attribute, column, and filter areas in Clarity and Relevance ticksheets.
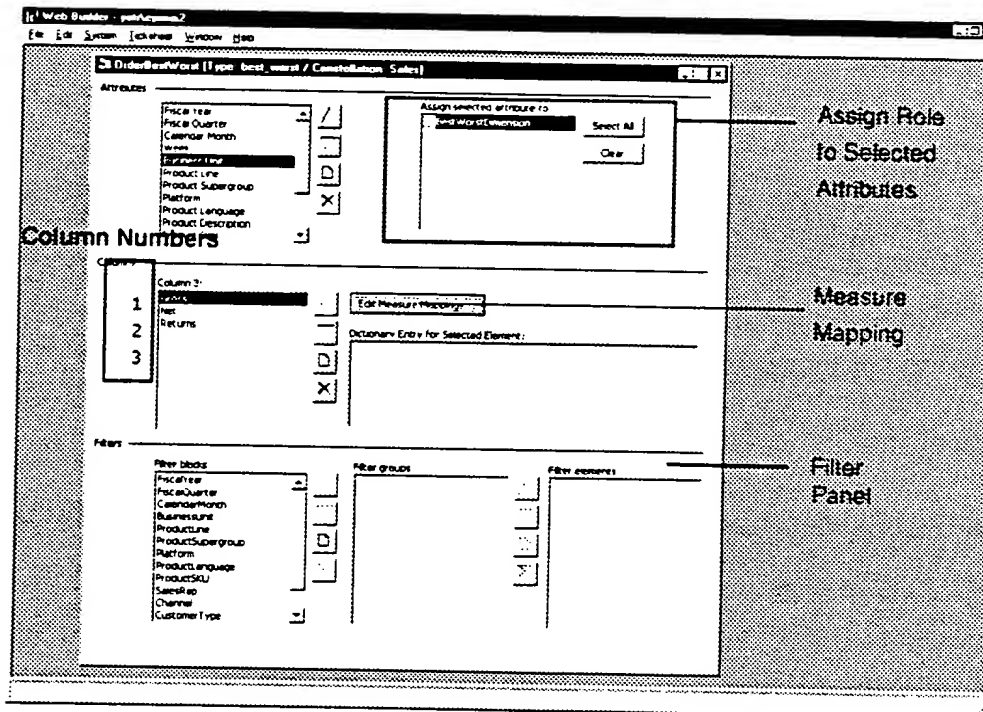
Figure 4-10 Ticksheet Dialog Box

Defining Attributes

Attributes from the dimension tables correspond to the columns and rows in a Clarity report. Attributes are also used as filtering criteria. You will use the Attribute Builder dialog box to add attributes to a ticksheet.

Follow these steps:

1. Click the New Item icon to the right of the Attributes list in the Ticksheet dialog box (or choose Attribute from the Attribute menu). The Attribute Builder dialog box shown in Figure 4-11 is displayed.

2. Enter the attribute's Label (the name that appears on the ticksheet).

3. Select the dimension column for this attribute from the base dimension table listing. (Right-click a plus sign to expand the tree.)

4. You may use the attribute as a link. For example, companies in a customer list could be linked to Web sites, such as http://www.co-select.com.

   Use the string val to indicate where the attribute is to be substituted; for example: www.val-select.com.

5. Click OK to add the attribute. It now appears in the Attributes box in the panel. Follow the same procedure to create (or modify) additional attributes.

6. After defining all of the attributes, you may select an attribute and assign it an attribute role, such as row only or column only for Clarity ticksheets. (The ticksheet type determines these options.) Use the Assign Selected Attributes area of the Ticksheet dialog box (see Figure 4-10, page 146).

   Important. Attributes can be defined only once per ticksheet. If you want a single attribute to appear in both the rows and columns listboxes of Clarity, you must assign it both of these roles: Clarity Row and Clarity Column.
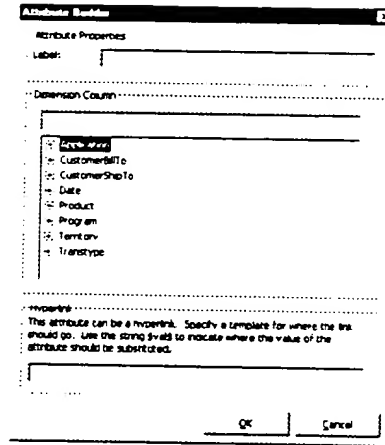
Figure 4-11 Attribute Builder Dialog Box

Defining Column Elements

The number of selection columns that you define for a ticksheet in Web Builder corresponds to the number that will display in the measurement section of a ticksheet (under the heading *For the measurements in each cell, select at least one item from each column below*).

A selection column consist of selection column elements. It is the combination of one element from each column that will be mapped to a single measure. The calculation of this single measure (applied to the data in the data warehouse) produces the values that displays in the cells of a Clarity or Relevance results page.

After using the Ticksheet Properties dialog box (see Figure 4-9, page 144) to enter the number of column elements, you can define elements for each column:

1. Click the column number (see Figure 4-10, page 146). (If elements have been defined, they will appear in the list.) Click the New icon to display the Column Elements Properties dialog box.

2. Enter the name for the selection column element. The label and abbreviation fields are automatically filled in with this same name, which you can edit.

3. Click the Dictionary Editor to add a dictionary entry. See *Defining Dictionary Entries* on page 142.

4. Click OK to add the element to the column.
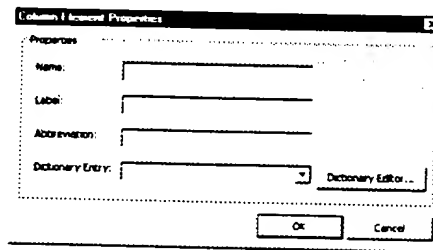
Figure 4-12 Column Element Properties Dialog Box

Measure Mapping

Measure mapping is the means by which the Epiphany system knows which measure to apply when a ticksheet user selects a set of column elements. As part of configuring a ticksheet, you need to inform the system which combination of selection column elements maps to a specific measure.

To set up measure mapping:

1. Click Edit Measure Mappings in the Ticksheet dialog box (see Figure 4-4, page 131) to open the ticksheet's Measure Mappings dialog box (Figure 4-13).

2. Select the selection column elements that comprise the measure from the drop-down list on the left.

3. Select the associated measure name from the listbox on the right. This measure is highlighted.

4. Continue to map each set of selection column elements to its associated measure.

   After you close this dialog box, this measure will be invoked whenever a user selects this combination of selection column elements in a ticksheet.

After you set up the mapping, you can click the Next (and Previous) buttons to cycle through all the combinations. Clicking a measure name (in the listbox on the right) displays the set of selection column elements that comprise it.

Note: Measure mappings must be set for each ticksheet. You may use the Data Copy panel of the Ticksheet Properties dialog box to import measure mappings from other ticksheets.

Elements
from each
selection
column

The measure
invoked when
a ticksheet user
selects this set
of elements

Measure Builder
defines calculations
(measure terms)
for a measure
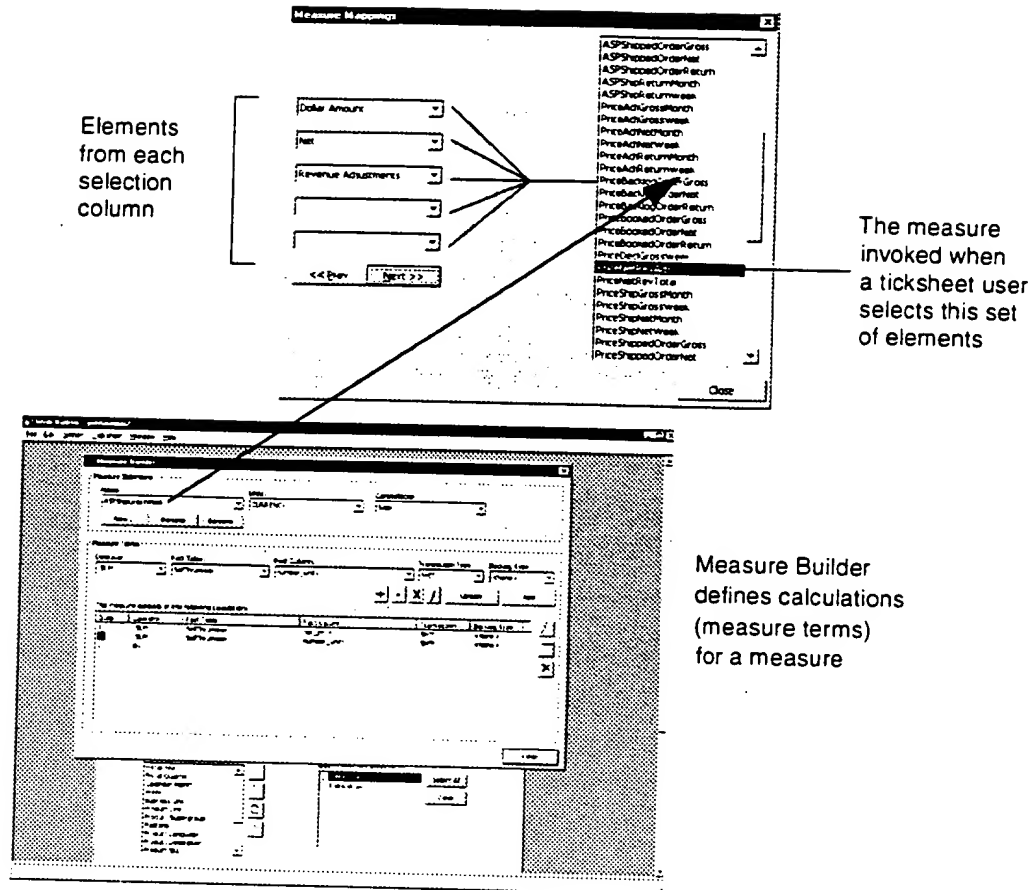
Figure 4-13 Measure Mapping

Defining Filters

Applying filters to a ticksheet query restricts the data that is accessed for that query. The Filters panel of the Ticksheet dialog box consists of Filter Blocks, Filter Groups, and Filter Elements that correspond to the same areas on a ticksheet. When you select an item in a Filter Block or Filter Group, the correct Filter menu displays in Web Builder's main menu.

There are three levels of filtering: the top, middle, and bottom level.

- Top level: A *Filter Block* is the attribute filter to be applied to the data, such as filtering the data by year, month, or customer. On a ticksheet, a Filter Block controls the user's ability to filter on a single dimension column (for a dimension role of the ticksheet's constellation). The Filter Block also defines the appearance of the filter (for example, check box versus listbox).

- Middle level: A *Filter Group* is a logical grouping of filter elements within a Filter Block. It is applicable only to check box Filter Blocks.

  For example, a Filter Block by year may have the Group Q197 (first quarter of 1997), with the months January '97, February '97, and March '97.

- Bottom level: *A Filter Element* is a single check box for filtering within a Filter Group, or a single entry within a listbox Filter Block.

To edit an item in a Filter Block or Filter Group, double-click it, which displays its Properties dialog box.

After you define filters for a ticksheet in Web Builder, they become options for the front-end user to select in the Filters window of a ticksheet.

Defining Filter Blocks

A Filter Block may be a text area, a check box, a static or dynamic listbox, or radio buttons. The contents of a dynamic listbox are updated when the Application Server starts. (Dynamic listboxes are refreshed while static listboxes never change.)

Use the Filter Block Properties dialog box to create a Filter Block:

1. In the Ticksheet dialog box, click the New icon for Filter Blocks, or choose Add Filter Block from the Filter Block menu. (Double-clicking an existing item in the Filter block area allows you to modify that item.) The Filter Block Properties dialog box is displayed.
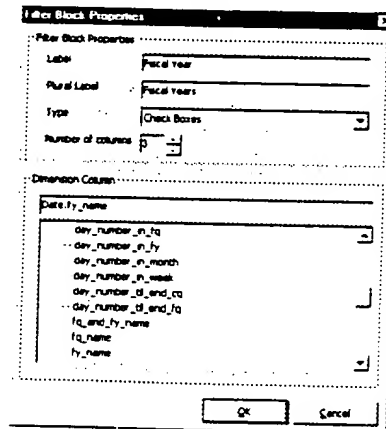
Figure 4-14 Filter Block Properties

2. In the Label textbox, enter the name for the heading on the ticksheet, such as Fiscal Year.

   Note: The database values are mapped to the label on both the ticksheet and the report, or results page.

3. In the Plural Label textbox, enter the plural of the label, which is what follows *All* as in All Fiscal Years on the ticksheet.

4. Select the Type from the drop-down list: check boxes, listbox (a static listbox), dynamic listbox, radio button, or text area.

5. For the check box type, select the number of columns that the attributes will be divided into; for example, Fiscal Quarter has four columns. For other types, this choice determines the height of the listbox or text area.

6. Select the dimension column that contains the attribute.

7. Click OK to add the Filter Block.

Defining Filter Groups

> *Important:* Grouping applies to check box Filter Blocks only. You cannot define a Filter Group for a filter unless it is a check box filter.

A sample check box Filter Grouping follows:

Group 1 — 1996: Q196, Q296, Q396, Q496

Group 2 — 1997: Q197, Q297, Q397, Q497

Group 3 — 1998: Q198, Q298, Q398, Q498

In all other cases (non-check box Filter Groups), a group is created automatically. This group's only purpose is to contain either the items in a listbox (a container for one group at a time), or the SQL for the query that makes up a dynamic listbox.

> *Note:* Check box Filter Groups provide a convenience for ticksheet users. Creating a Filter Group does not modify the database.

To define a Filter Group, follow these steps:

1. In the Ticksheet dialog box, click the New icon for Filter Groups, or choose Add Filter Group from the Filter Group menu. The Filter Group Builder dialog box is displayed (see Figure 4-15).

2. To enter names manually, enter a new item label (for the name on the ticksheet) and its value; for example; 1996 for the label and 1996 for the value. Click Add. The new item appears in the list as 1996 [1996].

3. To edit an item, simply click it. Its contents will be displayed in the New Item Label and New item value text areas for editing. Click Update to save any changes.
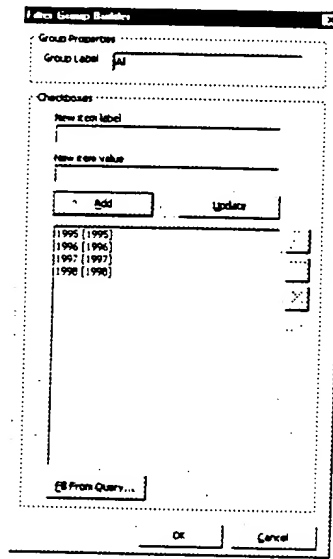
Figure 4-15 Filter Group Builder

4. To have the system fill in the list using an SQL SELECT statement (to extract a field from a table), click Fill From Query. The SQL Query dialog box (Figure 4-16) displays.
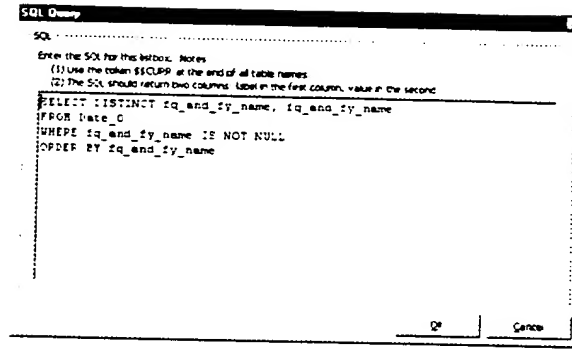
Figure 4-16 SQL Query Dialog Box

5. To have Web Builder create a template SQL for your filtering query, replace the column name and the table name for your data. Append the token `_0$$CURR` at the end of the table name; for example, `Product_0$$CURR`.

6. Click OK to begin the extraction.

The query provided by Web Builder has two columns by default: one for label and one for value.

Configuring Relevance Ticksheets

The configuration of a Relevance ticksheet is similar to that of a Clarity ticksheet. You can speed up the process by first creating a Clarity ticksheet, and then using the Data Copy panel of the Ticksheet Properties dialog box (Figure 4-5, page 134) to copy attributes, columns, and filters.

The only significant difference between Relevance and Clarity ticksheets is the attribute roles. Each Relevance ticksheet has its own attribute roles. The Best & Worst ticksheet has only one kind of role, a BestWorstDimension, and only attributes assigned this role appear in the ticksheet attributes area.

In general, there is no reason to conceal attributes from users. The exception is when there are so many that the ticksheet becomes cluttered, or when an attribute has extremely high cardinality (many values). For queries involving such attributes, the Best & Worst query may be very slow.

The Profiling ticksheet also has only one role, the ProfilingDimension. Again, only attributes assigned this role will appear on the ticksheet. Profiling is not effected by high-cardinality attributes as much as Best & Worst, so avoiding clutter is the main reason to not assign attributes to the ProfilingDimension role.

The Trends ticksheet has two roles, TrendsRow and TrendsColumn. The TrendsColumn role should be assigned only to time attributes, such as Fiscal Quarter, Fiscal Year, and so forth. The Trends Row role may be assigned to any non-time attributes.

The Quarter Projections ticksheet has two roles, ProjectionRow and ProjectionColumn. The attributes for ProjectionRow should be time attributes only, such as Fiscal Quarter or Month. The attributes for ProjectionColumn should be relative time attributes only, such as Days until End of Fiscal Quarter, or Weeks until End of Fiscal Quarter, or Days until End of Month.

CHAPTER FIVE

Epiphany Application Server

The Epiphany Application Server is the component of the Epiphany Application Suite that processes all user requests and returns query data in HTML format. All Epiphany applications, such as Clarity and Relevance, run on top of the Application Server. The Application Server has a multi-threaded design that allows several interactive connections to occur simultaneously.

In general, an Application Server interaction begins when the user points a browser at a Web URL address. The URL is first processed by the Web server (IIS 3.0), and then routed to the Epiphany proxy, **Epiphany.dll**. The proxy packages the request and sends it to the Application Server via a TCP/IP socket. The proxy waits for the Application Server to process the request and return a result. The proxy then takes the result and returns it to the user's browser via IIS. The proxy serves to separate the Application Server from the IIS process. This architecture allows several IIS machines to point at the same Application Server. The Application Server listens for requests, dispatches them to work threads, then processes the requests and returns the results. Figure 5-1 shows the role of the Application Server in the Epiphany system.

FRONT END

Web Browser

HTTP

Web
Server

Epiphany.dll

Epiphany
Application
Server

Java

BACK END

JDBC Driver

RDBMS
Server

Epiphany database
(EpiMart)

ODBC Layer

Source System   Source System   Source System
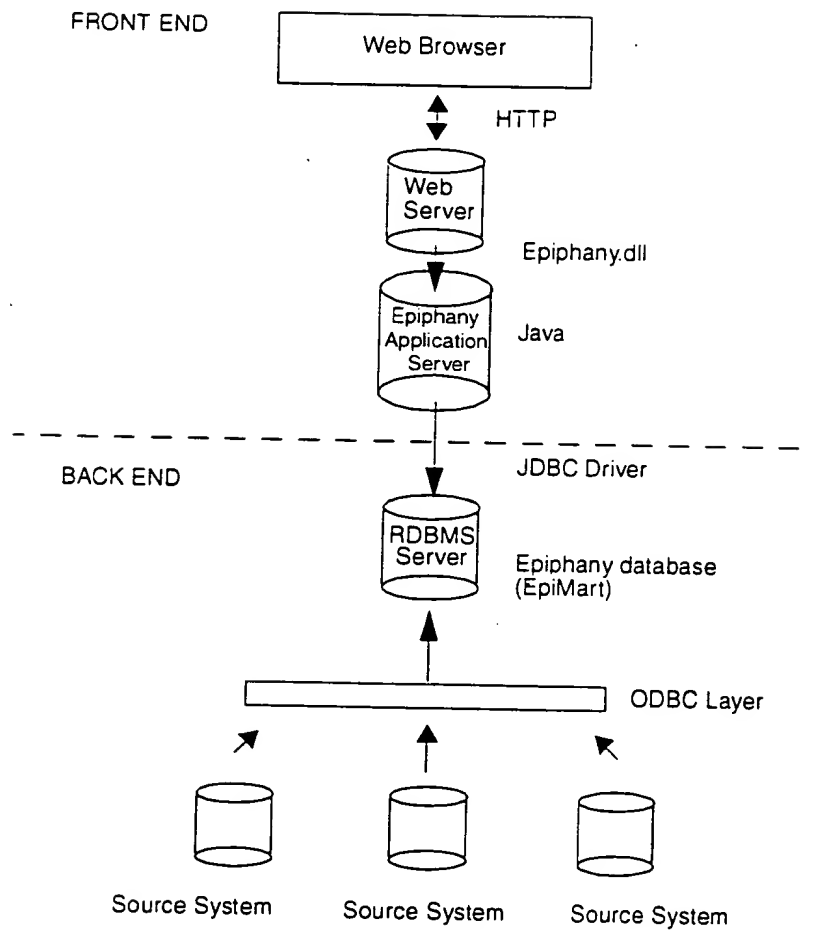
Figure 5-1 Epiphany System Diagram

The Application Server is a Java application that is normally run as an NT
Service. Java is not used on the client (browser) side.

This chapter covers the following topics related to the Epiphany Application Server:

For instructions on how to troubleshoot the Application Server, see Appendix I. This appendix also describes the Application Server error messages.

Starting and Stopping the Server

This section gives instructions for starting and stopping the Application Server when you run it as a service, or as a console application.

Running as a Service

A standard Epiphany software installation (as described in *Installing the Epiphany Software* on page 213) installs the Epiphany Application Server as an NT service. You can start and stop the service using the Windows *Control Panel\Services* menu. Follow these steps:

1. From the *Start* menu, choose *Settings\Control Panel\Services*.

2. Select the Epiphany Application Server. It is usually installed under the same name as the *instance_name* you specified during installation.

3. Click Stop to stop the Application Server.

4. Click Start to start it.

Determining if the Application Server Is Running

When you run the Application Server as a service, it may be difficult to tell whether it is running. To determine if the Application Server is running:

1. Check the NT Event log by going to the *Start\Programs\Administrative Tools (common)\Event Viewer* and opening the *Log\Application* menu.

   If the Service was started and is running, there will be an entry with Source = EpiAppServer. The message reads Epiphany Appserver <instancename> message: The Service was started.

2. Check for the existence of the Application Server's log file.

   The Application Server creates a log file immediately upon initialization, which will be named using the following convention:

   ```
   1998-06-08_14-48-15-32SRV.txt
   ```

where *1998* is the year, *06* is the month, and 08_14-48-15 is the day, hour, minute and second when the Application Server started running.

The log file resides in the logging directory, which is specified in the Windows Registry under the key:

HKEY_LOCAL_MACHINE\Software\Epiphany\Instances \instance_name *\SystemLogDir*

Normally, this log file is located under the directory: \instance_name *\web\WWWROOT\logfiles.*

If you have started the Application Server and it has been initialized, this log file resembles the one in Figure 5-2.

```
Adding session: __ global __|0
[EpiCenter] Initializing instance test0
   EpiMeta DSN : acme
   EpiMeta Username : sa
   EpiMart Database : Acme2_EpiMart
   Epimart Username : sa
   [TimeNav] Initializing instance test0
   [TimeNav] Initialized instance test0 in 2133 ms
[EpiCenter] Closing connections.
[EpiCenter] Initialized instance test0 in 11837 ms
AggNav initializing instance test0
AggNav initialized instance test0 in 1161 ms
Security manager initializing instance test0
Security manager initialized instance test0 in 130 ms
Save restore manager initializing instance test0
Save restore initialized instance test0 in 41 ms
[om] Beginning init in D:\TEMPL\
[om] Processed 47 templates.
Server Host: localhost
Server Port: 8081
Server Inst: test0
**** Epiphany AppServer test0 awaiting connections... ****
```

Figure 5-2 Sample Application Server Log File

Running as a Console Application

Usually, you will start Epiphany Application Server from the *Start* menu: *Settings\Control Panel\Services* menu. During debugging and the initial setup, however, it may be expedient to start the Application Server from a console window because all of the logging information is copied to the console.

You can manually start the Application Server by using the *jre* program (in your local path) from the directory:
C:\Program Files\Epiphany*instance_name*classes.

Enter the following command line:

```
jre -mx64M -classpath .;.\connect;.\connect.jar;.
    \EpiAppServer.jar;C:\PROGRA~1\
    JavaSoft\JRE\1.1\lib\rt.jar com.epiph-
    any.server.Server localhost:8081 @instancename
```

Notes:

If the **jre** program directory is not in your local path, you will have to pre-pend the pathname to the **jre** command. Normally, **jre** is installed in the *C:\Program Files\Javasoft\jre\1.1\bin* directory. This command example also assumes that the Application Server is running on the local machine on port 8081.

For security to function properly, the user who runs this command line must have certain NT user rights. Otherwise, no one will be able to log in (you will receive EpiLoginException violations).

When you manually start the Application Server using the **jre** command, a console window that echoes all of the logging information is displayed on your screen. The window must remain open while the Application Server runs. If you close this console window, the Application Server terminates. If you log out of your machine, the console window closes, and the Application Server terminates.

For Authentication to Work

For authentication to work, run the Application Server as a service under the Local System account, or under an administrator account that has special NT privileges. The special NT privileges are—

- Act as part of OS
- Increase quotas
- Replace a process level token

The Local System account already has these privileges, and almost always, installation will be set up to run the Application Server under the Local System account.

To run the Application Server from the command line, follow these steps:

Note: This procedure affects only the local machine, not the NT network.

1. Log in as a user who has local machine administration access, in addition to the special privileges listed above.

2. Assign special NT privileges to a user account. (You must be an administrator on the machine running the Application Server.) To do so, start User Manager and enter the local machine name in the Select Domain dialog box.

3. Choose Policies/User Rights from the menu.

4. Click Show Advanced User Rights and assign the account of the currently logged-in user to have the privileges specified above.

5. Reboot your machine for the privileges to take effect.

Command-line Arguments

The Application Server's main routine is located in the **com.epiphany.server.Server** class. The Application Server takes the following command-line arguments

```
jre -classpath classpath com.epiphany.server.Server
    machinename:port number @instancename DEBUG=1
```

If the machine name and port number are not specified, then they are read from the Registry under the *instance_name* directory. See *The Application Server's Registry Keys* on page 175 for more information. If the *instance_name* is not specified, then the server will read the following file to determine the default *instance_name* to use:

HKEY_LOCAL_MACHINE\Software\Epiphany\Instances\default key

The DEBUG=1 parameter will cause the Application Server to log extra information. This extra information includes the data about the "clean-up" routines that handle timed-out sessions and old command threads that have finished.

Refreshing the Application Server

Upon startup, the Application Server reads metadata and Registry information and caches it for use during normal query processing. The metadata that is read includes

- The Windows Registry
- The *config_master* table in the EpiMeta database
- Aggregate navigation information
- Time navigation information
- The templates used to render Clarity and Relevance result (report/chart) pages

- Security Information

- Ticksheet information

When any of the following events occur, you need to restart or refresh the Application Server:

- After an extraction, either the *current_datamart* has changed or there has been a change in a table that is used in a dynamic listbox filter.

- EpiCenter Manager was used to alter the EpiMart.

- Aggregates have been built or rebuilt since the last time the Application Server was started.

- Web Builder was used to add or modify a ticksheet, measure, or dataset.

- The Windows Registry entries under the following entry have changed since the last time the Application Server was started:

   HKEY_LOCAL_MACHINE/Software/Epiphany/instance_name

- EpiCenter Manager was used to change security permissions for a user or group.

> *Note:* The fact that a template file has changed does *not* require a restart or refresh of the server.

If you have determined that you need to refresh the Application Server, there are several ways to proceed. The simplest way to refresh the server is to stop and then restart the Application Server via the Services Control Panel. However, this requires manual intervention, and so would not be suitable for programmatic refresh (after an extraction, for example). This method also interrupts anyone currently using the system.

Another way to refresh the server is by using the **refresh.exe** program that was installed in your *installdir/win32* directory. This program sends a message to the Application Server instructing it to re-read all of the necessary information

mentioned above. It also re-parses all of the template files (in case the TemplateDir Registry key was changed) and re-initializes the Security objects.

Refresh will not interrupt the requests that are currently running on the Application Server. After the refresh has been completed, however, users who were previously logged on will need to log in again.This is because the security restrictions that affect all users' rights have been changed.

You may invoke the Refresh program via a DOS command-line or open the RefreshApp dialog box and enter command options (see *Invoking RefreshApp* for instructions).

The Refresh Command Line

The Refresh command syntax is:

refresh *localhost port username password*

where:

localhost   Specifies the name of the machine on which the Application Server is running.

port   Specifies the port number on which the Application Server is listening.

Username and *password*

Specify a valid NT account that has access to the Epiphany Application Suite.

These are the same values required for an end user to log into the top-level Epiphany Web page. In general, any account that is defined in the Security folder of the EpiCenter Manager application will work. See *Security* on page 109 for more information.

After the Refresh application has established a connection to the Application Server and sent the appropriate messages, it displays the following message and waits for a response:

```
Sent the REFRESH instruction to localhost
```

Normally, the Application Server takes about 30 seconds to refresh before returning an acknowledgment. (Times may vary based on the size of your EpiMeta database and the speed of your network, and other considerations.) Upon receiving this acknowledgment, the Refresh program displays:

```
Refresh [Build 3.2.0.1035]
--------------------------------------
Connecting to localhost:8081
Sent the REFRESH instruction to localhost
The refresh SUCCEEDED.
REFRESH operation took 18827 ms.
```

Otherwise, the program indicates that it has failed by outputting the response that it does receive, or by indicating that it has failed. For example, here is the ouput of a negative interaction in which the user/password failed.

```
Refresh [Build 3.2.0.1035]
--------------------------------------
Connecting to localhost:8081
Sent the REFRESH instruction to localhost
The refresh FAILED because the username/password combination was
invalid.
REFRESH operation took 3365 ms.
```

(The Refresh program always displays the amount of time that it has taken.)

Invoking RefreshApp

RefreshApp is the graphical user interface (GUI) version of **refresh.exe**. You may keep the RefreshApp window open and on your desktop during debugging.

Follow these steps to use the Refresh application:

1. Choose the Refresh application from the *Start* menu: *Programs\Epiphany\RefreshApp*.

2. For the AppServer name, enter the name of the machine on which the Application Server is running.

3. For the AppServer Port, enter the port number on which the Application Server is listening.

4. For the username and password, specify a valid NT account that has access to the Epiphany Application Suite.

    These are the same values required for an end user to log into the top-level Epiphany Web page. In general, any account that is defined in the Security folder of the EpiCenter Manager application will work. See *Security* on page 109 for more information.

5. Click Refresh to start the application.

    Messages display in the bottom of the dialog box that tell you whether the Refresh succeeded or failed (and the time that this event occurred).

6. In case of failure, click View Log to determine what caused the failure. The information that is normally printed to the screen during the execution of **refresh.exe** is displayed in this window.
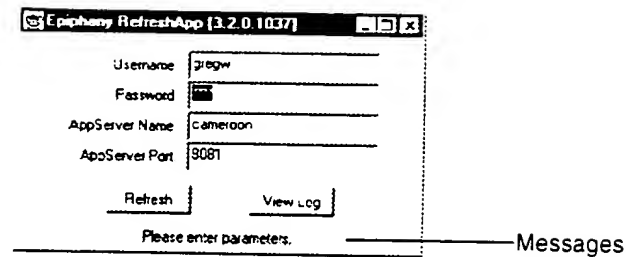
Figure 5-3 Epiphany RefreshApp Dialog Box

The EpiAppService Program

The EpiAppService program is the program that Windows NT uses to run the Application Server as an NT Service. You will use EpiAppService to—

- delete an Epiphany Application Server service.

- change the parameters of an Epiphany Application Server service.

- add a new Epiphany Application Server service; for example, create a service because the installation failed, or to install another instance.

Command Syntax

The **EpiAppService** command syntax is:

```
epiappservice -S servicename action
```

where:

-C

Creates a service. Registers a new service with the NT system. You must use the -E option with this option to specify the executable to use as the NT Service; for example:

```
EpiAppService -S "instance" -C -E "program name"
```

When you are configuring an Epiphany Application Server, the command line will look like:

```
EpiAppService -S "instance" -C -E "C:\Program Files\
Epiphany\instance\win32\EpiAppService.exe -S instance"
```

If the instance name has spaces in it, then you should use single quotes inside of the outside quotes. Do not use spaces within the instance name.

Run the EpiAppService program to create the service, which is also named EpiAppService. It is invoked with the -S option to specify the instance name. (The EpiAppService program is both the program that should be run as a service, as well as the program used to register the service.)

-E cmd

The command to associate with the service.

-DEP *service*	Allows you to specify a service on which the Application Server depends. The Service Manager will always load this dependency before loading the Epiphany Application Server. The service name supplied as an argument to this option must exactly match the service name used by the service in question, usually the Microsoft SQLServer.
-D	Deletes the service specified via the -S option from the NT Service Registry. When this service is deleted, only the entry in the NT Registry for the service is deleted; no executables are deleted. Use this to remove unused Application Server instances.
-G	The launch handler for the service. (Go.)
-T	sTarts a service. The installation program uses this option to start the Web server (which must be shut down during installation). It starts the service specified via the -S option, which is equivalent to clicking the Start button in the Control Panel\Services dialog box to start the desired service. You may use this option when you cannot access the Control Panels (for example, during RCMD or pcAnywhere remote access).
-P	StoPs the service specified via the -S option. (See the -T option.) The installation program uses this option to stop the Web server during the installation.
-K	Checks if the service specified via the -S option is running. The program returns a zero return code if the service is running, and a non-zero return code if it is not.

-?   Displays online help, a description of the command options.

EppiAppService Command Examples

Examples of the most common invokation of this program follow:

- Creation of a service because the installation failed, or you wish to install another instance. If the database server is on a different machine from the Application Server:

```
EpiAppService -S instname -C -E "C:\Program Files\
Epiphany\instname\Win32\EpiAppService -S instname"
```

     If the database server is on the same machine as the Application Server:

```
EpiAppService -S instname -C -E "C:\Program Files\
Epiphany\instname\Win32\EpiAppService -S instname" -DEP
MSSQLServer
```

- Deletion of an old service.

```
EpiAppService -S instname -D
```

The Application Server's Registry Keys

All Registry keys used by the Epiphany Application Server are located in

     HKEY_LOCAL_MACHINE/Software/Epiphany/Instances/instance_name

where *instance_name* specifies the name of the instance you entered during the Epiphany software installation. This directory contains the following keys:

AppServerHost   The name of the machine on which the Application Server is running. The default is the *localhost.*

Application ServerPort   The port number on which the Application Server is listening for connections. The default is 8081.

AppServerLogVerbosity This parameter specifies the degree of logging (verbosity level) that the Application Server performs. The default value is O. (For this release only the default value is supported.)

AppServerQueryTimeout

The number of seconds before the Application Server automatically terminates long-running queries. A query that requires more than this number of seconds to process will be terminated in order to prevent the exhaustion of system resources and run-away queries from bringing down the server.

AppSessionTimeout   The value in seconds for the lifetime of an idle session. If a session has been idle for this many seconds, then it is removed from the cache.

Build   The version number of the Application Server. This value should be 3.2.

ChartsOutputDir   The directory (full path) in which the *.epc* chart files will be stored.

DatabaseName   The name of the EpiMeta database.

DatabaseServer   The name of the database server on which the EpiMeta database is located.

DatabaseUsername   A username for logging into both the EpiMeta and EpiMart databases.

DatabasePassword   The password that corresponds to the DatabaseUsername account.

DatabaseType   Specifies the type of the database. This value should be *SQLServer6.5*.

Description	A textual description of the instance.
DSN	The name of the DSN that is used to connect to the EpiMeta database.
InstanceRootDir	The root directory into which the Epiphany Application Server has been installed.
ProxyLogFile	(Optional) Enables proxy logging. For example, setting *HKEY_LOCAL_MACHINE\SOFTWARE\Epiphany\ Instances\Stromboli\ProxyLogFile* to *C:\proxy.log* will create a *proxy.log* file if it does not already exist, and ·log information for every proxy submit to the Application Server. If an invalid path is specified in this ProxyLogFile Registry key, no logging will be performed.
SystemLogDir	The directory in which the Epiphany Application Server log files are written.
SystemLogDirWebpath	The name appended to http://machinename/*instance_name/* that informs the IIS Web server of the log files' locations.
TempDirGarbageLifetime	
	The lifetime in seconds of a temporary or log file created by the Epiphany Application Server. After this specified number of seconds from the creation date, a temporary file, log file, or *.epc* chart file will be erased by the Temporary FileManager in the Application Server. Garbage collection occurs at Application Server startup and periodically when there are free cycles.

TemplateDir

The directory in which the Epiphany Application Server templates are stored.

Driver

The JDBC driver used to connect to an EpiMart database. The default is **connect.microsoft.MicrosoftDriver**. *Do not change this value.*

SecurityClass

(Optional) The Java class to use for security authentication. If it is not specified, the **com.epiphany.security.EpiNTLogon** class is used.

The Epiphany Proxy

The Epiphany proxy (**Epiphany.dll**) is an ISAPI application that mediates the requests and responses between the IIS Web server and the Epiphany Application Server. All user requests for Epiphany pages are directed to the proxy http://machinename/scripts/*instance_name*/Epiphany.dll.

This proxy bundles the request into a package that conforms to a strict Epiphany format and sends the package to the Epiphany Application server through a TCP/IP socket. The proxy uses the Windows Registry to find the Epiphany Application Server. In particular, the proxy parses the *instance_name* out of the requesting URL. It then opens the AppServerHost and AppServerPort with that instance's Registry tree, and uses this information to connect to the Application Server. The Application Server processes the request and sends a result back to the proxy. The proxy displays the result in the user's browser.

Note: The *instance_name* in the URL must match the *instance_name* as defined in the Windows Registry (configured from the installation program). This allows one proxy to direct requests to several different Application Server instances. This is of value when you want to have two versions, such as a release and a test instance.

Proxy Logging

The **Epiphany.dll ISAPI** proxy supports rudimentary logging. If you suspect that the proxy is not passing all parameters to the Application Server, you can enable logging of all parameters that the proxy submits.

Proxy logging, which is optional, is a diagnostic tool for identifying problems, not a run-time logging facility. Because the log file may grow arbitrarily large if proxy logging is always enabled, use it only in the event of a suspected problem with the **Epiphany.dll** proxy.

An example of a proxy log file:

```
    Processing new request. Header: mGET q p v0.8 oHTTP/1.0 nzhenya
P80 r192.0.0.147 aMozilla/4.04 [en] (WinNT; I ;Nav) u U/scripts/
stromboli/Epiphany.dll S/scripts/stromboli/Epiphany.dll Requested
dispatched. Request data length was 0

      Processing new request. Header: mPOST q p v0.8 oHTTP/1.0
nzhenya P80 r192.0.0.147 aMozilla/4.04 [en] (WinNT; I ;Nav) u U/
scripts/stromboli/Epiphany.dll S/scripts/stromboli/Epiphany.dll
sEP70SER=emile&EP_PWD=zhenya&EP_TMPL=toplevel&EP_APPLICATION_TYPE=Clarity
Requested dispatched. Request data length was 72
```

The log file consists of blocks, with each block representing a submit to the proxy DLL. Each new submit starts with `Processing new request. Header:` and is followed by the header. The header consists of header items, each beginning with a letter that describes the type of item, followed by the value of item.

The first letter of the header item is defined below.

m	method POST or GET
q	query string (used with GET)
v	version
o	HTTP protocol
n	server name
P	server port
r	remote IP address
a	user agent
u	user name (if Web server is using authentication)
U	URL
S	script name

The header will be followed by the data section if the submit was of the type POST. First, a number of bytes is printed (172), then the actual data, which should be the same size as the number of bytes printed. Finally, if the submit was successful, the following line is printed:

```
Requested dispatched. Request data ....
```

If the submit failed, then the log file should read:

```
FAILED TO DISPATCH REQUEST DUE TO A SOCKET ERROR.
```

Application Server Logging

Most components of the Application Server maintain a log file of their activities. This section describes each of these logs, their directory location, and their diagnostic use.

The components that generate logs and the contents of those logs are briefly described below (and in more detail later in this section).

Server	The **com.epiphany.server.Server** class generates a log file of connections made to the Application Server. This file contains the time at which a connection was accepted, the number of that connection, the time at which the request was finished, the total time required by the request, and certain messages printed to the log during the processing of that request. The latter category includes the session ID of the connection, the template used to process that request, the number of bytes generated in the response, exceptions that might be generated during the processing of the request, or the user name used to log into the server.
SecurityManager	The **com.epiphany.security.SecurityManager** class generates a log that contains the users and groups in the system.
SaveRestore Manager	This creates a log that contains all of the SQL statements executed for Save/Restore purposes. Calls to certain functions such as **getReportKey()** and **loadSaveRestoreQuery()** are also logged with their return values.

Clarity or Relevance Query Log

> Each Clarity or Relevance request generates a log that contains the submitted ticksheet's parameters, information about how the ticksheet was parsed, all of the SQL statements executed during the processing of the request, and information about how long the processing took.

Log File Location

In the Windows NT Registry on the Application Server machine, there is a Registry key named *HKEY_LOCAL_MACHINE/Software/Epiphany/Instances/* instance_name/*SystemLogDir* that specifies the location of all log files generated by the Application Server.

Log File Naming Conventions

All log file names begin with a prefix that indicates the date and time when the log file was first created. The format is

```
YYYY-MM-DD_HH-MM-SS-MSname.txt
```

where YYYY stands for the year, MM for the month, DD for the day, HH for the hour, MM for the minute, SS for the second and MS for the millisecond. *name* can be one of the following: SRV, SECURITY, SAVE_RESTORE, or QM_*sessionid*. The logs of queries for Clarity and Relevance applications have the suffix QM_*sessionid*.

The Server Log

When you start the Application Server, the server object creates a log immediately after the Registry has been opened and read. The Registry must be opened first because it contains the *SYSTEMLOGDIR* Registry key that specifies exactly where the log files are stored.

```
**** Epiphany AppServer Acme awaiting connections... ****
Mon Jun 08 14:48:50 PDT 1998: +++++ Accepted [1] @ [897342530272]
----- Dispatched [1] @ [897342530322]
Free/Total Memory: 165064/3297272
Setting to default template (no app): toplevel
No session exists/created. Displayed login screen.
4865 bytes generated.
        ----- Finished [1] @ [897342530773]
501ms


Mon Jun 08 14:48:59 PDT 1998: +++++ Accepted [2] @ [897342539215]
        ----- Dispatched [2] @ [897342539245]
Free/Total Memory: 719832/3342328
Template passed in: toplevel
Got EP_USER: lslater
Adding session: EPIPHANY\lslater|192.0.0.12
5184 bytes generated.
        ----- Finished [2] @ [897342539615]
400ms
```

Note that the server writes this information for each socket connection it accepts:

```
Date: +++++ Accepted [connection number] @ [milliseconds since
    epoch]
```

For example:

```
        Sun May 03 17:54:16 PDT 1998: +++++ Accepting [1] @
[894243256100]
```

The *Date* is written using the **java.lang.Date.toString()** method. Hence, it uses the default formatting on the server machine. The connection number is the

numerical ID of the connection that was accepted. Numerical IDs are assigned in increasing order, starting from 1. The *time in ms since epoch* is determined using the **System.getCurrentMillis()** method. It represents the number of milliseconds that have elapsed since January 1, 1970.

During the processing of the request, certain messages might be printed to the server log if no other log is available. This is the case for the TemplateServer and the EpiAdminServer.

After the request has been started, the server indicates the end time of the request in this format:

```
Finished [<connection number>] @ [milliseconds since epoch] 400ms
```

The Security Manager

The Security log first prints out all of the groups recognized by the system. Each entry in this first section contains the group name, the key in the database, and a list of ticksheets accessible by members of the group. For example:

```
Groups: {Administrators=

Group: Administrators / key: 4 / world save query access: 1

Access List:

Assigned ticksheets:

ExecutiveBestWorst,ExecutiveProfiles,ExecutiveProjections,
    ExecutiveTrends,Expense,
```

The next section of this log file contains a list of all the users recognized by the system. Each entry contains a user name, the groups to which the user belongs, and the list of ticksheets that the user has permission to view. Note that the ticksheet list is mostly empty since all users belong to at least one group, and therefore inherit ticksheet permissions.

Save and Restore Manager

The Save and Restore Manager logs function calls made to **getReportKey()** and **loadSaveRestoreKey()**. All of the SQL executed during the retrieval/storage of a query is also logged to this file. Each entry is prefixed by the date and time of the entry. For example:

```
Sun May 03 18:55:26 PDT 1998: loadSaveRestoreQuery(Sales
        Projections, sarah)
```

Clarity and Relevance Applications

> *Note:* If the **log** link at the bottom of an HTML Clarity/Relevance result page does not take you to a log, make sure that the SytemLogWebDir *Registry* key refers to an alias that has been configured on the Web server. Also make sure that the directory and files have correct read permissions.

Each Clarity or Relevance application creates its own log file before performing any non-trivial processing. The log contains the version of the Application Server and the date/time of the request. The second line begins with `APPLICATION parameters:` and subsequent lines contain all of the name-value pairs submitted to the Application Server for this request. Multiple values submitted with the same name are shown separated by commas. A blank line follows, and then the Application filters, dimensions, and measures that could be parsed from the submitted ticksheet's parameters are logged.

Next, the system logs all of the SQL statements needed to compute the answer to the request, the amount of time in milliseconds to process the query, and the number of rows returned for all SQL queries.

The log ends with the CLEANUP message.

Application Server Security

Authentication is performed outside of the Epiphany system, which does not capture password information. This external authentication requires an authentication module. For example, the NT authentication module authenticates users with an NT domain controller. The Security Manager inside the Application Server loads the authentication module specified through the SecurityClass Registry key at initialization. Each authentication module supports a fixed API that includes methods to authenticate users, add new users to the Epiphany system, and sync-up outside information on the user (such as group memberships) with Epiphany metadata.

Currently, Epiphany provides two authentication modules: the NT authentication module EpiNTLogon, and an insecure authentication module called EpiPassThruLogon. Use EpiPassThruLogon only for testing and debugging. Optionally, you can add an LDAP authentication module, X.500 module, and other similar modules.

The optional Registry key SecurityClass under the *instance_name* Registry directory controls which security module is loaded. You must specify the full class path to the security module. If this key is omitted, the system uses the default security module **com.epiphany.security.EpiNTLogon**, which uses NT to perform user authentication.

The means by which the authentication information (username and password) reaches the Application Server is as follows. Users log into the Epiphany system either through a login template or through a Web server authentication mechanism, such as Basic Authentication or NTLM.

When the Application Server receives the user name and password, it calls the Security Manager to log in the user. The Security Manager loads an authentication module, and attempts the login process. If the process is successful, depending on the authentication module, one of the following steps is taken:

- If the user exists in the EpiMeta database, user group memberships outside of the Epiphany system will be synched up with group memberships in the EpiMeta database.

- If the user does *not* exist in the EpiMeta database, but is authorized to use the Epiphany system, then a user account will be created in the EpiMeta database. User group memberships outside of the Epiphany system, such as NT, will be synced-up with group memberships in the EpiMeta database. For example, assume your EpiMeta had a group named EPIPHANYUSERS configured with access to all of the ticksheets. If a user name Joe (a valid NT user who belongs to the NT group EPIPHANYUSERS) supplies a correct password, then Joe will be automatically added to the EpiMeta metadata as a user who belongs to the Epiphany group EPIPHANYUSERS.

Only group memberships for a group whose Group Definitions dialog box in EpiCenter Manager has the Synchronize option selected will be synced-up. Sync-up occurs if:

- the user who logs in is 1) a member of a Group X outside of the Epiphany system; 2) Group X exists inside the Epiphany system; and 3) the user is not a member of Group X inside the Epiphany system. Sync-up will create a group membership to Group X for this user inside the Epiphany system.

- the user who logs in is 1) a member of a Group X inside the Epiphany system, but 2) the user is not a member of this group outside of the Epiphany system. Sync-up will remove a group membership from the Epiphany system.

If the sync-up process requires a creation of a new group membership for a user, certain access rights are set on this membership. The ability to save queries has the access rights of Save Group/Default, and global-level save access is Inherit. (See *Security* on page 109 for more information.)

For a group to be the same in EpiMeta and the NT domain, it needs to have the identical name (case sensitive) in both. The group name in the NT domain includes the domain name, such as *Epiphany\Sarah*. You need to make sure that the group name in the EpiMeta includes the domain name for that group.

When the user is authenticated and user information is synced-up, Security Manager determines if the user is authorized to use the Epiphany system. The user is so authorized if he or she belongs to at least one group in the Epiphany system. (The user must also must log in with a valid password.)

> *Important* An authenticated user is authorized to use the Epiphany system if and only if that user is a member of at least one group in the Epiphany system after sync-up.

Authentication Modules

The following authentication modules are included with the Epiphany software release 3.2.

Module	Class Name
NT (default)	com.epiphany.security.EpiNTLogon
Pass through	com.epiphany.security.EpiPassThruLogon

- NT (default)

  NT domain authentication. NT groups are imported into the Epiphany system through EpiCenter Manager. As mentioned, these groups need have the Synchronize option selected in the Group Definition dialog box, which means that memberships to those groups will be synced-up. Users

who belong to those groups can log into the Epiphany system. When a user logs in, his or her NT group memberships are synced-up to the EpiMeta database. It is also possible to create Epiphany-only groups inside EpiCenter Manager by simply not selecting the Synchronize option. Thus, the Epiphany administrator may create arbitrarily complex permission hierarchies using EpiCenter Manager, independent of the manner in which NT domain security is set up.

* Pass through (*for in-house debugging purposes only; should never be used at a customer site after the initial Epiphany system setup*)

This is an insecure authentication that has no password. This is an Epiphany-only development module that ignores NT authentication all together. You do not need passwords in this model: specify your user name exactly as it appears in EpiCenter Manager (it must include a domain name if such exists), and you will be logged in. There is no sync-up process. That means that an authenticated user that does not exist in the EpiMeta database will not be allowed to use the Epiphany system. Epiphany groups and users are created and managed through EpiCenter Manager.

Authentication Module Tips

* If you want to use NT-related authentication modules, make sure that **EpiNTLoginJNI.dll** is in your system path.

* Users that do not belong to any groups in the Epiphany system are not allowed to log in. An error message that says that user is authenticated but not authorized to use Epiphany system is displayed whenever an unauthorized but NT-authenticated user logs into the Application Server. User memberships may be adjusted upon login if the user is a member of synchronized groups in Epiphany system. A user must be a member of at least one Epiphany group after the synchronization process completes.

- The optional Registry key SecurityClass (located in the *instance_name* Registry directory) controls which security authentication module is loaded. The full class path to the security module must be specified as the value for this key. If this key is omitted, the default security module is **com.epiphany.security.EpiNTLogon**, which uses NT to perform user authentication.

- The **Toplevel.template** is the template that appears immediately after a user logs into the Epiphany system. The name of this template is configurable with the an optional ToplevelTemplate Registry entry in the *Instances* directory. You can load the **company.template** instead of **toplevel.template** by substituting your company name in `ToplevelTemplate=`*company*.

- Depending on how IIS security is set up, one of the following situations occurs upon login:

  - If Allow Anonymous authentication is enabled, the login dialog box is displayed when the user attempts to use the system for the first time, or the user session times out. It is almost always sufficient to specify the username only. The domain name is located automatically.

  - The search order that authentication uses to find the user account is as follows. First, the authentication mechanism looks in the local machine's Security Access Manager (SAM), then it checks with the primary domain controller, and afterwards checks with trusted domains. If there are multiple users with the same name between domains and/or a local machine that runs Application Server, specify the full user *name-domainname\username*, or the *local_machine_name\username* if the user logs in from a local machine's Security Access Manager (SAM).

  - If Basic Authentication is enabled, the browser displays a login dialog box when the user attempts to access the Epiphany

system for the first time. However, when the user's session times out, no re-login is required. The user is automatically logged in again, and a new user session created.

- If NTLM authentication is enabled, Internet Explorer automatically performs authentication of the user without displaying a login dialog box, although Netscape displays it. If Basic Authentication or NTLM is on, the login dialog box does not appear in the Web browser.

Important: Do not create your own domains. Doing so introduces multiple domains, with the Epiphany machine in one domain and the user accounts of the Epiphany system in another domain. In order for the authentication module **com.epiphany.security.EpiNTLogon** to work properly, a two-way domain trust between the Epiphany domain and the customer domain is required.

If you set up a new domain for the machine that runs the Epiphany Application Server, set up a two-way trust and name the machine and the domain differently. In general, do not use the same string for domain names, machine names, and user names.

- The following applies to Sychronized groups:

EpiNTLogon requires the NT domain for lists of global and local groups. The names of these groups will be matched to the names of the groups in the EpiMeta. A group name will be matched if its domain name and group name match. Therefore, group *foo* will not match to group *EPIPHANY\foo*. (The match is case insensitive for both group name and domain name.) If a user is a member of an NT group *EPIPHANY\foo* and EpiMeta has a group called *epiphany\FOO*, there will be a match.

  - If a user is a member of a local group and the group has a global group as a member, the global group will not be picked for synchronization process. Only the groups for which the user is an immediate member are considered for synchronization process.

- If a user logs in with an account that is local to the Application Server machine, then a membership to a special group called None is automatically retrieved from the machine's SAM. (Every user in the local SAM has a membership in a special global group called None although this group does not exist on the machine.) For this reason, do not create synchronizable groups called None in EpiCenter Manager.

- Avoid having the same name for domain names, machine names, and user names. For example, if the Application Server runs on the machine *foo*, and the user called *foo* attempts to log in, access may be denied. If the Application Server is running on a machine named *foo*, and a user named *foo* logs in from the primary domain, or from the local Security Access Manager (SAM) of the machine *foo*, authentication will succeed. If user *foo* logs in from a trust domain, however, the authentication will fail. The only way to log in as *foo* from another domain is to give the full name for the user account upon login: *domainname\foo*.

Configuring Output Templates

Application Server output results from a template-processing mechanism. Template files are files parsed by the system and used as templates for the output that the Application Server produces. For example, all of the static HTML and JavaScript that is a part of every Epiphany-produced ticksheet resides in a template file.

A template file contains three different kinds of text: plain HTML, a special environment variable tag, and a special class invocation tag. These are described below.

Plain HTML

Most of the template file contains plain text, Java script code, or HTML that is copied verbatim when the output is generated. This allows a system administrator to make small modifications to the look and feel of certain pages. For example, one can change the **clientlogo.gif** that appears in the upper right-hand portion of the Web page.

A special environment variable tag

The `<!-EP ENV="{name}"-->` tag is an environment variable tag that is parsed and textually substituted with a string from the environment hash. The Environment hash is a structure that is maintained by the Application Server during the processing of a query. It contains name-value pairs meant to be accessible from a template file. When this special tag is parsed, `{name}` is used to query the Environment hash for a value string. The string is then printed in place of the tag. The tag can contain special instructions for the encoding of the value string. For example `<!-EP ENV="foo" QUOTEESCAPE -->` forces the Application Server to encode all of the < and quotes in the value string as `>` and `"`. There is also a URLESCAPE that encodes the value string as per the x-www-form-urlencoded MIME type. See the online MIME standards RFC 2045 through RFC 2049 for more information. The URL is www.oac.uci.edu/indiv/ehood/MIME/MIME.html.

A special class invocation tag

The class invocation tag

```
<!-ENV CLASS="com.epiphany.presentation.... "-->
```

instructs the Application Server to load dynamically the class specified as the CLASS parameter, and to invoke the *execute* method of that class. Any text that is produced by that class is substituted in place of the tag. Most of the dynamic content of an HTML result page is generated by a class invocation. For example, the table of results on a Clarity result page is generated by the class:

com.epiphany.presentation.HTMLTableOutputMethod

To support localization, class invocation tags also support two auxiliary parameters, LANGUAGE and COUNTRY. These parameters localize the text that is produced by the class.

For example, in all of the templates shipped with the installation program, you will find the following method at the bottom of the page. This method displays the date and time at which the output was produced.

```
<!-EP CLASS="com.epiphany.presentation.DateMethod"
LANGUAGE="en"-->
```

Although you may make simple HTML changes to the template files, do not change the environment variables (see *Environment Variables* below), or the class invocation tags. The environment variables allow you to substitute installation-specific or query-specific information into the result pages. For example, the name of the machine on which the Application Server is running can be accessed by the EPV_URL_PREFIX environment variable. This information is required for any hyperlink or drill-down request that requires communication with the Application Server to work. The alternative would be to hard-code the machine name into templates.This, however, requires tedious

and error-prone editing to be done at every installation site and every time that a machine name changes. In some cases, it is impossible to hard-code a fixed value into the template.

For example, after a user logs in, he or she is assigned a unique session ID that is embedded in each of the ticksheets that the user sees. The session ID is generated randomly on the server at the time the user logs in, so there is no way to hard-code this information into the template."

Environment Variables

The environment variables available for customer use are described below.

APPSRVR_QUERY_TIMEOUT

> The same value as the Application ServerTimeout Registry key.

EPA_ANCHOR

> Used when someone clicks a hyperlinked item in a ticksheet to display its dictionary entry.

EPV_APPSERVER_VERSION_STRING

> The version string for the Application Server, for example, 3.2.1029.

EPV_BESTWORST_ATTS

> Used in Relevance Best & Worst ticksheets as a list of attributes.

EPV_INSTANCE_NAME   The name of the instance.

EPV_URL_PREFIX

> The URL of the machine. This value is parsed from the initial URL that is used to access the Application Server.

EP_SES

> The session ID of the current session. Used in place of the *username/password* after a user has logged in.

EP_TMPL   The template that will be used to format the output.

EP_USER   The username of the user logging in. This value is only
                   available when the user initially logs in.

EXTENDON_SELECT   A special purpose parameter used in the profiling
                   machinery.

QUERYLOGFILENAME   The name of the query log file. It is defined for all
                   Clarity and Relevance queries.

ERRORMESSAGE   When the system encounters an error, this variable
                   contains the text that describes the error condition.

Customizing Error Messages

You may customize most of the Application Server's error messages via the file *errormessages_en.properties* located in the *installdir\Classes* directory, where *installdir* is the directory in which you installed the Application Server. The prefix *_en* indicates that these are English-language messages. In future releases, the Application Server will support error messages in multiple languages.

The text that displays in the Application Server log when the server receives an exception corresponds to the text entered in this file for each anticipated exception.

Every unique exception generated by the Application Server has a name-value pair line in the *errormessage_en.properties* file. For example:

```
>--------------

com.epiphany.exception.EpicenterInvalidValueException=Invalid value
found in the metadata: {0}.\nWhile executing the following sql
statement, the Epicenter loader encountered an invalid value. Please
make sure that the epicenter has been built properly, and that it has
not been corrupted.\nSQL Statement:\n{1}

com.epiphany.exception.EpiQueryInvalidMeasureException=You have
selected a measure that is not defined in the metadata for this
ticksheet.\n{0,choice, 1#The Measure you selected was|1< The Measures
you selected were}:\n{1}\nGo back to the ticksheet and choose another
measure.

>--------------
```

The name-value pair appears on one line. If your messages are cut off, make sure that the there are no line breaks between the text. If you need to insert new lines in your message text, use \n.

Briefly, each exception has a function that makes the appropriate calls to read the exception text from the *errormessage_en.properties* file and to process it by substituting the correct strings. The macro expansion capability is simple. The parameter {n} refers to the nth argument passed to the exception. The parameter

```
{n, choice, i#blah blah|k< blek blek}
```

indicates to print either blah blah or blek blek based on the nth argument to the exception.

- If n==i, then it is blah blah.

- If k<n, then it is blek blek.

(You cannot include brackets ({ }) in the exception text.)

This architecture—

- separates the text describing the error from the state of error.

- facilitates localization of error messages.

- allows you to change the error text without having to recomplile.

Name Replacement

The file *dictionary_en.properties*, located in the *installdir\Classes* directory, allows the system administrator to replace dynamically occurrences of words with any arbitrary sequence of words.

The structure of the *dictionary_en.properties* file is exactly the same as the *errormessages_en.properties* file. The file consists of name=value pairs on single lines. For example, a file with the following line:

```
Fiscal Year=L'ann\u0431e fiscal
```

substitutes the string L'année fiscal for the string Fiscal Year in all of the output pages. The name=value pairs can contain escaped Unicode characters. The standard way to escape a Unicode character is to use /uXXXX where the XXXX represent the hexadecimal representation of a character's code. By default, no name replacement is performed. If the *dictionary_en.properties* file is not present, then the Application Server displays this status message for each query:

```
Dictionary for en not found.
```

To remove this message, create a dummy *dictionary_en.properties* file that contains no characters.

See Appendix I for a description of the Application Server error messages and conditions.

Epiphany Application Server

APPENDIX A

Installation

This appendix covers the Epiphany software installation procedures.

Epiphany System Requirements

Before installing the Clarity and Relevance 3.2 software, please ensure that your system meets the following specifications:

- An Intel-based NT Server 4.0, Service Pack 3, IIS 3.0 (Web server), and Microsoft SQLServer 6.5 with Service Pack 3 or later.

- A minimum of 128 Mb of RAM; 256 or 512 Mb is preferred. The machine that runs the Epiphany Application Server requires a minimum of 64 Mb.

- One or two CPUs (two are preferred) with a speed of 200 or greater MHz.

- An adequate amount of disk space. The Epiphany-specific software requires about 8 Mb. You will also create two new, empty databases: one for EpiMeta (metadata) and the other for EpiMart (your data warehouse).

  The recommended size for EpiMeta is 50 Mb. for data and 50 Mb. for logs.

The size you allocate for your EpiMart depends on the amount of your customer data. If you already have specifications for your EpiCenter and know the number of fact rows and dimension rows, you can use the following rough estimate in which the number of bytes of space equals:

```
2 * (10 * #fact rows * (100 + #bytes user-fact-data)
+  #dimension rows * (20 + #bytes user-dimension-data))
+ 300 Mb
```

This calculation is based on two copies (A vs. B) of the database and aggregates that are approximately ten times the base table size, with additional space added for staging tables, indexes, and keys. *Aggregates* are pre-calculated summaries stored in the data warehouse to accelerate queries.

As part of the Epiphany software installation, you will need to allocate at least 200 Mb for the temporary database *tempdb,* and 100 Mb for the log database.

- A RAID disk array is preferred although not necessary. RAID-0 is rated as having the highest performance; RAID-5 emphasizes data integrity.

- The computer must be connected to a network and have TCP/IP installed.

- Microsoft Exchange messaging client software must be installed for e-mail notification regarding the status of the database extraction process.

Notes:
When you install the Epiphany software, the first screen displays your system configuration.

Multiple network card configurations are not currently supported.

Installation Overview

The overall steps for installing the Clarity and Relevance 3.2 software involve—

- installing Microsoft Windows NT Server 4.0 (and its service pack; the service pack installs the version of the IIS server you need).

- installing Microsoft Access (one of the Microsoft Office programs).

- installing Microsoft SQL 6.5 Server (and its service pack).

- configuring Microsoft SQLServer.

- installing Clarity and Relevance 3.2 software from the Epiphany CD-ROM.

- setting up the Epiphany Application Server.

The Epiphany software does not have to be installed on the machine that has Microsoft SQLServer software installed. You also have the option of installing for remote administration.

> *Note:* If applicable to your site, you will need to create custom DSNs (ODBC Data Source Names) that are required by connectors.

The complete installation procedure as documented here takes approximately two hours.

These instructions are directed towards consultants and database administrators who are familiar with installing software for the Windows NT operating system. If specific instructions are not given, the default values are acceptable.

If you have any questions about your network, please consult your local network system administrator. For third-party applications, please refer to the documentation that came with your software for more information.

Installing Windows NT Server 4.0

Follow these steps to install your Windows NT Server 4.0:

1. Insert Disc 1 of the Microsoft Back Office 2.5 CD-ROM into your CD-ROM drive, open the CD-ROM, and double-click the **Setup.exe** file.

2. The Welcome to NT Server Setup screen displays. Press Enter. Follow the instructions on the screen as they pertain to your system.

3. Select the directory location to install the Windows NT Server 4.0. (The default is acceptable.) The files are copied to your computer.

4. After the files have been copied, remove the CD-ROM and restart your computer as indicated.

5. When the Windows NT Setup screen appears, begin the Windows NT Setup.

Windows NT Setup

For the Windows NT Setup:

- Assign a computer name of 15 or fewer characters and write this name down for future use.

- Indicate the server type as a stand-alone server.

- Enter your administrator account name and password.

- Create a repair disk in the event of an emergency.

- Select components (the default values are acceptable).

Windows NT Networking

If your computer is part of a network, the Wired to Network field displays the network name and the Microsoft Internet Information (IIS) server (the Web server) is automatically installed. The network settings must be configured after consulting with your local network system administrator (see Step 4).

To set up Windows NT networking:

1. Select Start Search for network adapters.

2. TCP/IP as protocol is selected (and cannot be deselected).

3. Accept the default selections for Network Services and click Next to continue.

4. Depending on what your network system administrator specifies, either select DHCP or enter your TCP/IP address.

5. In the TCP/IP Properties dialog box, enter your TCP/IP address, the Subnet Mask, and Default Gateway (as assigned by your local network system administrator).
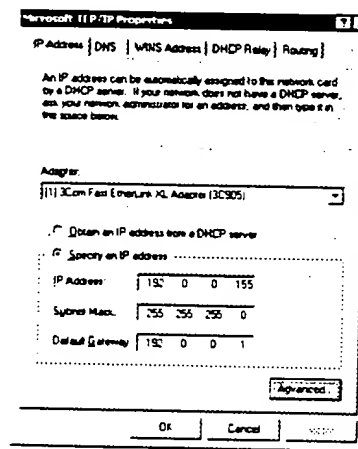
Figure A-1 TCP/IP Properties Dialog Box

- Click the dialog box's DNS tab and enter the domain name.

- Click the WINS Address tab. In this dialog box, enter the address for the primary WINS server. Click Apply in the WINS Address dialog box, and then click OK.

- For Show bindings, select all services from the check box list. Normally, the defaults are acceptable.

- Do not click Enable unless a red international symbol for NO (red circle with slash through it) appears next to a service. This will not appear on a clean install.

- Click Next.

- Start the network by clicking Next.

- In the Workgroup/Domain dialog box, enter the domain name in *Create computer account in the domain* ONLY if this is appropriate for your site.

This completes the Windows NT Setup. Next you will set up the Internet Information Services.

Microsoft Internet Information Server (IIS) Setup

Note: Windows NT Setup installs IIS 2.0; the Service Pack 3 upgrades IIS 2.0 to IIS 3.0.

As part of the Windows NT Setup, after you finish setting up the network parameters, the Finishing Setup dialog box is displayed, which informs you that you will now start setting up MS IIS 2.0. Follow these instructions:

- Use the default values for the Options dialog box.

- You may specify different values in the Database Publishing Directory dialog box. If the security features of IIS are to be used, the *wwwroot* directory must reside on an NTFS file system.

- Select the Microsoft SQLServer driver to be installed.

- Set the Time Zone.

This completes the Windows NT installation. You may remove the CD-ROM and restart the machine.

Note: After installing the new operating system, you may have to install drivers for special graphics cards or other hardware.

Install the Service Pack

The last step of the Windows NT installation procedure is to install the Microsoft Windows NT Service Pack 3. The Service Pack provides upgrades to Windows NT 4.0 and installs the IIS 3.0 Web Server, replacing the 2.0 version installed during the Windows NT installation.

Follow these steps to install the NT Service Pack:

1. Log on to Windows NT as the administrator.

2. Insert the Service Pack 3 CD-ROM into your CD-ROM drive.

3. Click Install Service Pack on the screen (or run **spsetup.exe** from the CD-ROM). The Service Pack Setup dialog box displays.

4. Because this is a clean install, you do not want to create an uninstall directory.

5. Click Finish, remove the CD-ROM, and restart your computer.

Install Microsoft Office

Microsoft Office may be installed from the Microsoft Office disk.

1. Insert this CD-ROM into your CD-ROM drive and follow the instructions on the screen.

2. Accept all of the defaults.

Installing Microsoft Office installs Microsoft Access, which installs the database used for importing and exporting metadata.

Installing Microsoft SQLServer

You will install SQLServer and its utilities from the Microsoft Back Office CD-ROM, Disc 2.

Follow these steps:

1. Insert the CD-ROM Disc 2 into your CD-ROM drive and open the disk on the desktop.

2. Open the Sql65 folder and then open the i386 folder. Double-click Setup.

3. Select Install SQLServer and Utilities.

4. For SQLServer's installation path, use the default—if there is enough space.

5. For the Master device creation, use the default—if there is enough space.

6. In the Installation Options dialog box (Figure A-3) click the Sets button next to Character Set, which displays the Select Character Set dialog box. Select 850 Multilingual (see Figure A-2) and click OK.
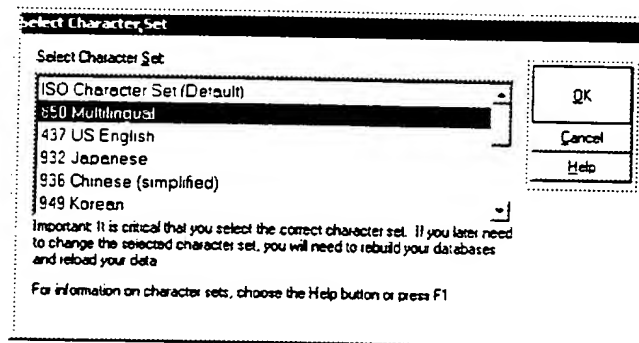
Figure A-2 Select Character Set Dialog Box

7. In the Installation Options dialog box, you may use the default Sort Order.

8. In this same dialog box, click the Networks button. For additional network support, select Named Pipes and TCP/IP Sockets in the Select Network Protocols dialog box (Figure A-3).

9. Select the check boxes Auto Start SQL Server at boot time and Auto Start SQL Executive at boot time (so these start whenever the Windows NT server boots).

10. After completing the Installation Options dialog box, click Continue.

11. When asked for the SQL Execution Log on Account, enter your password.

12. For the TCP/IP Socket Number, use the default port number.

Now you are ready to install the Service Pack.
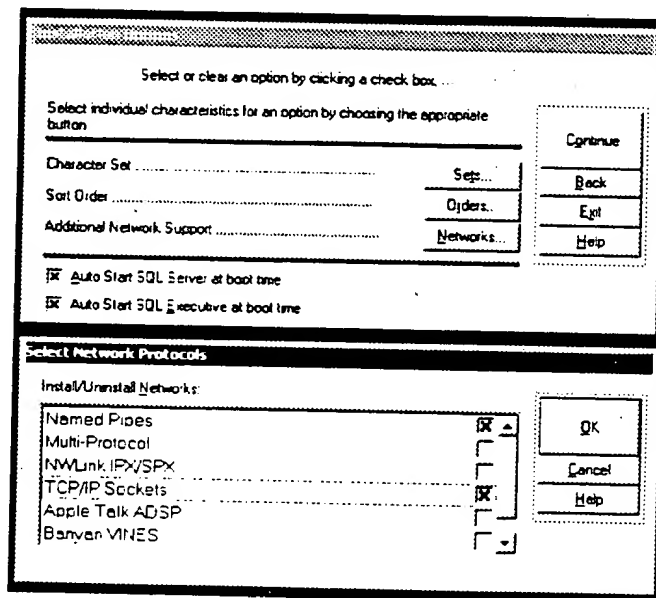
Figure A-3 Installation Options: Select Network Protocol

Install the Service Pack

Install the Microsoft SQLServer Service Pack 3 to upgrade SQLServer.

- Insert the CD-ROM into your CD-ROM drive and follow the instructions on the screen.

- After installation, remove the CD-ROM and restart your computer.

Microsoft SQLServer Setup

Follow these steps to set up SQLServer version 6.5:

1. Log in and start the SQL Enterprise Manager.

2. Select Server Register Server from the menu. Specify **sa** as the login ID and leave the password field blank.

3. Enter the machine name in the server box.

4. Click Register.

Setting Up Your Databases

Use the SQLServer Manager to set up new databases (see Figure A-5). You need to configure disk space and database space for two new, empty databases: one for your EpiMeta (metadata) and one for your EpiMart (data warehouse). The recommended size of your EpiMeta is 100 Mb; 50 Mb for data and 50 Mb for logs. (See *Epiphany System Requirements* on page 201.)

Right-click the Database Devices directory. Select New Device from the pop-up menu. The New Database Device dialog box (Figure A-4, page 211) is displayed.

To set up one of your two databases:
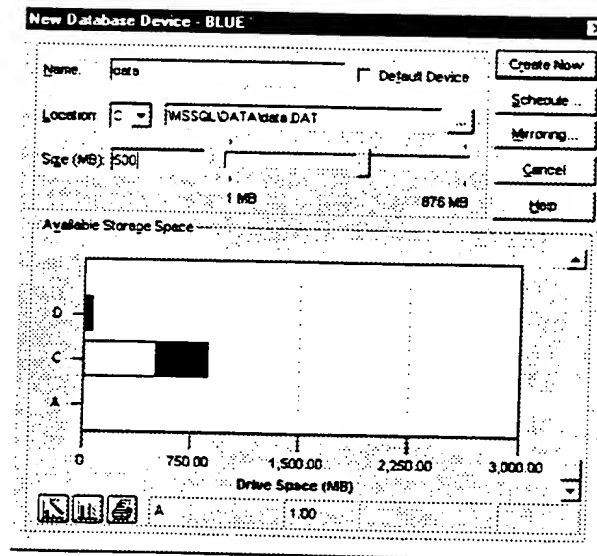
Figure A-4 New Database Device Dialog Box

1. Enter the name for the database device. The naming your chose for your EpiMeta should distinguish it from your EpiMart; for example, *Corp_EpiMeta* and *Corp_EpiMart*.

2. Select the location where there is enough disk space available and enter the size in megabytes. Click Create Now.

   The size needs to be large enough to store the EpiMeta or EpiMart database for your site.

   You may modify this size at any time.

3. Select *tempdb* in the database tree and right-click it. Select Edit from the pop-up menu. In the Edit dialog box, click Expand.

4. In the Expand dialog box, enter the size to expand the temporary database, in addition to the device where it is to be expanded (the recommended size is 200 Mb).

5. Click the Expand button. The *tempdb* will be expanded to the indicated size.

Repeat this procedure to create your other database.

SQLServer Configuration

Using the Server Manager, right-click the server you just installed in the Microsoft SQLServers tree; see Figure A-5. Select the Configuration tab in the Server Configuration/Options dialog box. Change the following values as indicated:

- *Locks.* Set the value to 20,000.

- *Memory.* Assign as much memory as your system configuration allows. For example, 100,000 2 Kb blocks equals 200 Mb of memory.

  The value that is displayed is a theoretical maximum, *not* the maximum level that the machine will handle. If you set the memory value higher than a value your machine can run, SQLServer will not restart.

- *Open objects.* Set the value to 15,000.

Server
Manager
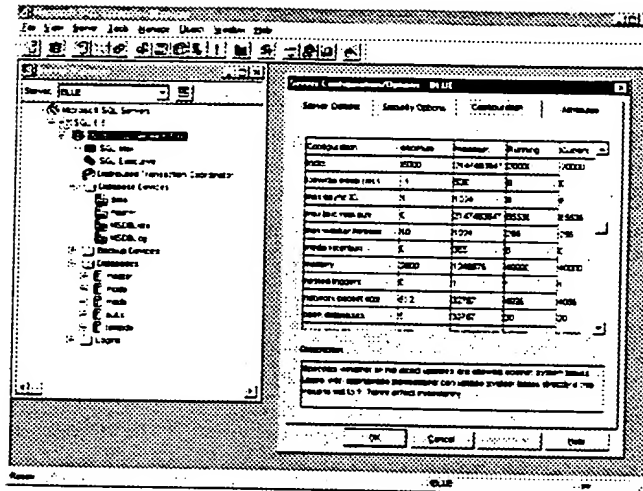
Figure A-5 Server Manager and Server Configuration/Options Dialog Box

Installing the Epiphany Software

There are two types of installation:

- *Remote Administration* installs the programs needed to remotely administer an Epiphany system. This package installs Web Builder and EpiCenter Manager (**EpiMgr**), Refresh, and the extraction tools necessary to administer an EpiCenter database (in addition to the other DLLs and libraries required by these programs).

- *Application Server,* which installs all of the components for Remote Administration mentioned above, plus the Application Server. The Application Server is the component of the Epiphany Application Suite that processes all user requests and returns query results in HTML format.

Depending on your system configuration, you may be restricted to the Remote Administration setup. If so, the installation screen provides instructions.

important: The Microsoft Java virtual machine needs to be installed on your system. If it is not, you will be prompted to exit the installation program and install it. This software is included with Internet Explorer 4.0, which is available on your installation media (\I386\Utils\I.E 4.0I).

To install the Epiphany software:

1. Exit all open Windows applications, and if this is a re-install, or an install over a previously installed system, make sure that the Epiphany Application Server has been stopped.

   From the Start menu, go to *Settings \Control Panel\Services* and manually stop the Application Server. If you do not, the DLLs that are copied by the installation program will not be copied correctly.

2. Insert the Epiphany CD-ROM into your CD-ROM drive.

3. Double-click **Setup.exe** on the CD-ROM.

4. If the Microsoft Java virtual machine is not installed on your system, you are requested to exit this installation and install it.

5. The System Configuration screen displays your system's configuration data (for your information).

6. Select the type of setup: *Application Server* or *Remote Administration.* Follow the instructions below for your type.

Application Server

A Services window shows which services will be stopped before the file copy begins.

1. Enter the instance name. This is the name of the service as it appears in the Service Control Panel.

   An instance name distinguishes among multiple installations of the Epiphany software on the same machine. Typically, you will install the software once on a machine and thus have one instance. You are asked if this instance will be the default instance. If so, the **Setup** program makes this instance the default Web server destination.

2. Enter the name of the database server and the database name. The database name is the name of the EpiMeta database. This database does not need to exist yet. (The Registry keys are populated based on what you enter.)

3. Enter the username/password for the database server. The user must have system administrator (SA) privileges.

4. Select the destination directory for the Epiphany software. The default is *C:\Program Files\Epiphany*instance_name.

5. Enter the location of the run-time reports and charts (ticksheets) that end users will access. *C:\Program Files\Epiphany*instance_name*Charts* is the default. Create a new folder as requested.

6. Select the components to be installed: Epiphany System Programs, Applications Server Runtime files, Applications Server Java Classes, and on-line documentation in Adobe Acrobat/Reader PDF format.

   If you want to install only the Applications Server, select Applications Server Runtime files and Applications Server Java Classes (and the documentation); otherwise, select all of these to be installed on your machine.

7. Select the default program icon location.

This completes the Epiphany software installation. Next, go to *Setting Up the Epiphany Application Server* on this page.

Remote Administration

1. Select the destination directory for the Epiphany software. The default is *C:\Program Files\Epiphany*instance_name.

2. Enter the location of the run-time reports and charts (ticksheets) that end users will access. *C:\Program Files\Epiphany*instance_name*\Charts* is the default. Create a new folder as requested.

3. Select the components to be installed: Epiphany System Programs and on-line documentation in Adobe Acrobat/Reader PDF format. For the Remote Administration, do not select the Application Server components.

4. Select the default program icon location.

   This completes the Epiphany software installation.

   Note: If you want to use the NT Performance Monitor to monitor the extraction process, go to page 222.

Setting Up the Epiphany Application Server

The Epiphany proxy is the only component of the Epiphany Application Suite that interacts with IIS. For the Epiphany Application Suite to function properly, set the IIS configuration values according to the instructions in this section.

The Epiphany Application Server requires sufficient memory (at least 64 Mb) to function properly. The machine that runs the Application Server needs to be set to 256 colors mode.

The steps you follow depend on whether you are using IIS 3.0 or IIS 4.0 (see *IIS 4.0 Settings* on page 219).

IIS 3.0 Settings

You will use the Internet Service Manager dialog box to configure these settings.

1. From the Start menu, select *Start\Programs\Microsoft Internet Server (Common)\Internet Service Manager.*

2. Right-click your server icon in the Internet Service Manager window. The WWW Services Properties dialog box is displayed.

3. Select the Directory Security tab. In the Anonymous Access and Authentication Control panel, click Edit.

4. The dialog box that displays has tabs for Services, Directories, Logging, and Advanced, which need to be set as described below.

   Note: These are the default settings.

- Services Tab

  TCP Port = 80

  Connection Timeout = 90

  Maximum Connections = 100000

  Anonymous Login Panel values:

  - Username

    An NT username that has file access permission for all of the Epiphany installed files and directories. This username also needs permission to read the Epiphany entries in the Registry; that is, all entries under the *HKEY_LOCAL_MACHINE\ Software\Epiphany* directory in the Registry.

  - Password Authentication Panel values:

    The password for this username.

Password Authentication

- Select Allow Anonymous Login.

- Directories

Select the directory name in the listing.

Click Edit Properties in the Edit Properties dialog box.

Select Virtual Directory.

Enter the alias name for these directories in the text box.

Directory	Alias
C:\Inetpub\scripts	\scripts
C:\ProgramFiles\Epiphany\ instance_name\web\wwwroot	\instance_name

Allow read and execute access. Click OK.

Configure the following two fields:

- Select Enable Default Document. Enter this default document name: **default.htm**

- Select Directory Browsing Allowed.

- Logging Tab

You may use the default IIS settings, which are logging enabled, log to files, logging from standard, automatic open, log daily, log file directory: *C:\WINNT\System32\LogFiles.*

- Advanced Tab

Select Granted Access.

IIS 4.0 Settings

Note: For the Clarity and Relevance 3.2 release, Epiphany has not thoroughly tested IIS 4.0. Preliminary tests, however, have not indicated any compatibility problems.

To configure Applications Server settings for IIS 4.0:

1. Open the IIS 4.0 Service Manager from the *Start* menu by choosing: *Programs\Internet Service Manager.*

2. Right-click your server icon and select Properties from the pop-up menu. The Properties dialog box is displayed in the Master Properties panel. Select WWW Services as the Master Properties. Click Edit. In the WWW Service Master Properties dialog box, select Directory Security. In the Anonymous Access and Authentication Control Panel, click Edit. The Authenticate from Methods dialog box is displayed. Select Allow Anonymous Access. Click Edit.

3. Configure Anonymous Login with file access permissions for reading and writing to all Epiphany files.

   Anonymous User Account:

   - Username

     An NT username that has file access permission for all of the Epiphany installed files and directories. This username also needs permission to read the Epiphany entries in the Registry; that is, all entries under the *HKEY_LOCAL_MACHINE\ Software\Epiphany* branch in the Registry.

     Make sure that the directories that contain the **Epiphany.dll** and the **makechart.dll** files have execute permissions.

   - Password

     The password for this username.

4. Using the Virtual Directory tab, set up aliases for the following two directories:

Directory	Alias
C:\Inetpub\scripts	\scripts
C:\ProgramFiles\Epiphany*instance_name*\ web\wwwroot	*instance_name*

Manual Chart Install

The Epiphany charting solution consists of two modules: an ISAPI **DLLmakechart.dll** and a COM object **gsmaker.exe**. You need to install the charts on the machine that runs the AppServer. This machine needs to be running in 256-color mode.

Before proceeding with this installation, stop the Web server and kill the **gsmaker.exe** process if one is already running. Stopping the Web server is necessary to unload the **makechart.dll**. If you are unable to stop the Web server or kill the **gsmaker.exe** (using the Task Manager or **kill** utility), reboot your machine.

Place **makechart.dll** in the same directory as the *www* directory for the instance. If there are multiple instances present on the machine, you need **makechart.dll** in each instance *www* directory. (You can copy **makechart.dll** to the destination.)

Note that **makechart.dll** requires the following Registry entries under *HKEY_LOCAL_MACHINE\Software\Epiphany\Instances\instance_name.*

- ChartsOutputDir

  The physical directory where the **.epc* charting files are stored. These files are used to display charts, and are generated whenever charts are requested. By default, these files are removed daily by the Temporary File Manager. To set the number of seconds between Temporary File Manager runs, change the Registry key *TempDirGarbageLifetime.*

- SystemLogDir

    The directory where the *charts.log* file is to be placed. This log file logs every call to the **makechart.dll**.

The **gsmaker** executable is a Microsoft COM object that needs to be registered. If **gsmaker.exe** was already installed on this machine, then make a backup of the old executable and copy a new **gsmaker.exe** over it. Since the old COM object is registered, you do not need to re-register the new one.

If, however, you are installing **gsmaker.exe** for the first time, you need to register it as follows:

1. Copy **newgsmaker.exe** to
    C:\ProgramFiles\Epiphany\instance_name*win32*, and change to that directory.

2. Run the following commands (which are case sensitive):

    ```
    gsmaker  -UnregServer
    gsmaker  -RegServer
    ```


The **gsmaker.exe** process appears in the Task Manager if the registration was successful.

To verify whether **gsmaker.exe** is registered properly:

1. Run **dcomcnfg.exe** from the command line.

2. Select Application EpiChart Class.

3. In the General tab, make sure that the **gsmaker.exe** entry points to the right location. In the Identity tab, make sure that Launching User is selected.

4. In the Security tab make sure that the Everyone special group has access and launch permissions to **gsmaker.exe**.

>If custom permissions are set, click the Edit button to check permissions. If default permissions are selected, close the EpiChart Class Properties dialog box and open the Default Security tab to check the permissions.

The **gsmaker** program uses the Graphics Server Charting Package to draw the charts. You need to add the following Graphics Server files to the *C:\winnt\system32* directory if they are not already present:

- **gsgif32.dll**

- **gsprop32.dll**

- **gsw32.exe**

- **gswag32.dll**

- **gswdll32.dll**

This completes the Application Server setup. If you want to use the NT Performance Monitor to monitor the extraction process, after setting up your Application Server, go to the next section.

Setting Up NT Performance Monitoring for Extraction (Optional)

Performance monitoring of the extraction process is optional. Although every **extract.exe** is instrumented using the standard NT Performance Monitoring facility, you need to take these steps to use the NT Performance Monitor to monitor an EpiChannel job's progress:

1. Your *Epiphany*\instance_name*win32* installation directory must contain these items: **EpiPerfMon.dll**, **EpiPerfMon.ini**, and **EpiPerfMon.reg**.

2. **EpiPerfMon.reg** has values specified for this Registry key: *HKEY_LOCAL_MACHINE\System\CurrentControlSet\ Services\Epi\Performance*

Running **EpiPerfMon.reg** enters this key and its values into the NT Registry. Before doing so, you may edit this file so that the value *Library* points to the correct path where **EpiPerfMon.dll** resides, that is, your current *Win32* directory.

Edit **EpiPerfMon.reg** to change the *instance_name* to the correct value for the *Library* path, save the file and exit.

Note: Because most Epiphany files are installed as read-only files, you need to change permissions before editing.

3. Run **EpiPerfMon.reg**. To check the results, open the Registry and navigate to the key specified in Step 2. Ensure that the path supplied for *Library* is correct. If it is not, the *Epiphany Channels* object entry does not appear in the object list for the Performance Monitor.

4. From a command line, go to the *Win32* directory of the current instance and run:

```
unlodctr Epi
lodctr EpiPerfmon.ini
```

where `unloadctr Epi` unloads parameters from any previous Registry entry.

EpiPerfMon.Ini initializes the values associated with *Epi.*

Note: Epi is the default driver and key specified in **EpiPerfmon.reg**, which is described in *Monitoring Jobs* on page 53.)

You are now able to start the NT Performance Monitor and monitor the progress of EpiChannel. See *Monitoring Jobs* on page 53 for more information.

After Installing the Epiphany Software

After installing the Epiphany software you can:

- Run the Epiphany tools. such as EpiCenter Manager to configure your EpiCenter and to set up jobs for extracting source system data. Instructions are given in Chapter 3 of this *Guide*. Please read Chapters 1 and 2 for background information before attempting to use EpiCenter Manager.

- Enter your e-mail password to receive notification of Epiphany system status. You will use the Configuration dialog box in EpiCenter Manager to set this up. Instructions are given in Chapter 3.

- Use the Web Builder program to design ticksheets. Instructions are given in Chapter 4.

- Run the Epiphany extraction program (EpiChannel) to extract data from your source systems and place them in the Epiphany database (the EpiMart). EpiChannel performs the jobs you defined in EpiCenter Manager.

APPENDIX B

Epiphany Macros

This appendix describes the Epiphany-supplied system calls and SQL macros.

System Call Macros

For the most part, you will use Epiphany system call macros for the transfer of information related to the locations of files and database logins from the EpiMeta database to system calls. Many systems calls such as **mv** and **mkdir** are unable to read a database, and few system calls will be able to read Epiphany-created structures. Rather than leaving each system call to its own means to determine its appropriate information, EpiChannel may pass this information to the system call on the command line.

System Call Macro Syntax

The syntax for system-call macros can be either

```
$$VERB[arg1, arg2, ...]
```

or

```
$$VERB
```

Unless stated otherwise, the arguments are the names of roles defined with the job. You may use multiple arguments in most cases, which results in an

expansion of each argument. Note that the expansion of both of the following is the same:

```
$$verb[arg1, arg2]

$$verb[arg1] $$verb[arg2]
```

If the verb does not match one of the special words, no expansion of that verb occurs; for example:

```
echo $$notAverb
```

expands to

```
echo $$notAverb
```

If the argument is a role name, and the job does not define that role, the system call is considered to have an error.

Table B-1 describes the Epiphany system call macros.

> *Note:* The Usage column in the following table stands for expected frequency of use.

Table B-1: Epiphany System Call Macros

Macro	Purpose	Usage	Definition
AGG	Child Procs	High	The name of the Aggbuilder executable in $$EPIBIN. For example: $$AGG $$EXC_ARGS -j $$JOB_NAME
AGGVERIFY	Child Procs	Low	The name of the **aggverify** executable in $$EPIBIN.
APPSERVERHOST	Registry	Low	The value of this Registry variable.
APPSERVERPORT	Registry	Low	The value of this Registry variable.
CHARTSLOGFILE	Registry	Low	The value of this Registry variable.
CHARTSOUTPUTDIR	Registry	Low	The value of this Registry variable.

Table B-1: Epiphany System Call Macros

DATABASE	Database Login	High	Translates to the name of the database or instance.
			For example:
			
			isql /S $$SERVER[Tests] /U
			    $$USER[Tests] /P
			    $$PASSWORD[Tests] /d
			    $$DATABASE[Tests] /w 300 /i
			    /Q "gen_tests_run"

DBVENDOR	Database Login	Medium	Translates to the vendor of the database technology.
DEBUG_LEVEL	Command Line	Low	Translates to the current verbosity level of EpiChannel. Use this to pass EpiChannel's verbosity onto the subprocesses it spawns via system calls.

DIRNAME	File ID	Medium	Translates to the directory name of the data store, without the last filename component and without a trailing slash. If the role is *working dir*, the directory name's last component is a unique subdirectory generated for this particular run of EpiChannel. For example:
			
			    echo DIRNAME is
			    $$DIRNAME[Working Directory]
			
			(but with no arguments it is $$DIRNAME).

DSN	Database Login	Medium	Translates to the ODBC connection string for the database. This string may be generated even for databases accessed using native API's.
EPIBIN	Child Procs	Medium	The name of the *win32* directory under the *InstanceRootDir* Registry variable.
EXC	Child Procs	High	The name of the **extract** executable in $$EPIBIN.
EXC_ARGS	Child Procs	High	The recognized portions of the **extract** command line.

Table B-1: Epiphany System Call Macros

EXC_CMD	Child Procs	Medium	The **extract** program and its arguments other than job name. You can use this to fire sub-**extract** runs. For example: SSEXC_CMD -j performance
EXCVERIFY	Child Procs	Low	The name of the **excverify** executable in SSEPIBIX.
FILENAME	File ID	Medium	Translates to the filename of the data store. If the role is *Working Dir.* the file name is the name of the EpiChannel log file. For example: echo FILENAME is SSFILENAME[Working Directory] (but with no arguments it is SSFILENAME).
INSTANCE_NAME	Registry	Medium	The name of the instance's Registry subtree.
INSTANCEROOTDIR	Child Procs	Low	Value of the *InstanceRootDir* Registry variable.
ISS	Database Login	Low	Translates to a number associated with this source of information.
JOB_NAME	Child Procs	High	The name of the current job.
PASSWORD	Database Login	High	Translates to the password associated with the database login.
PATH	File ID	Medium	Translates to the directory name and file name placed together (in the DOS file path format).
PROGRAM_NAME	Child Procs	Low	The name of the current **extract** program.
REGISTRY_EPIPATH	Registry	Low	Name of the Epiphany Registry key.
REGISTRY_ROOT	Command Line	Low	Translates to the Registry key used by EpiChannel.
SERVER	Database Login	High	Translates to the database host server name (the machine's name) in the case of Microsoft SQLServer. or the SQLNet ID (sid) in the case of Oracle. SERVER and SQLNET are identical in behavior and can be interchanged.

Table B-1: Epiphany System Call Macros

SQLNET	Database Login	High	Translates to the database host server name (the machine's name) in the case of Microsoft SQLServer. or the SQLNet ID (sid) in the case of Oracle. SERVER and SQLNET are identical in behavior and can be interchanged.
USER	Database Login	High	Translates to the username associated with the database login.
VERSION	Database Login	Low	Translates to a release number or string associated with this database's installation.

Epiphany SQL Macros

Major database vendors have developed proprietary extensions to SQL that customers may choose to use in their SQL code because of their features. Epiphany also provides SQL macros for SQL Epiphany products. These macros, which are available as an option to customers, attempt to be database vendor-independent.

Epiphany supplies SQL macros to resolve these issues:

- Support coding of SQL statements that works with the syntax of multiple database engines.

- Support SQL that adapts to the structure of the star schema as defined via the EpiCenter Manager.

- Support extraction of contiguous but non-overlapping subsets from those tables that have dates or ascending identification fields.

Vendor-independent Macros

Although it is possible to avoid proprietary extensions to SQL, the features they offer may be necessary. As a result, consumers of SQL inherit problems raised by these differences unless they decide to bind themselves to a single database vendor. (For example, a function called NVL in one database is equivalent to a function called ISNULL in another database.)

Epiphany supports multiple database vendors, and Epiphany-executed SQL statements are a major consumer of SQL features. The Epiphany vendor-independent macros serve to provide some degree of isolation from database vendor differences. Epiphany uses these SQL macros in its own SQL so that it does not need to support different "codelines" of SQL.

Although these macros are available to use by Epiphany customers, their use is optional. Customers concerned only with a single database vendor might avoid all use of these macros. However, customers who use multiple vendors now or plan to in the future may choose to use the Epiphany SQL vendor-independence macros in the SQL statements they create.

SQL Macro Usage

An example of both preferred and less preferable SQL macro usage follows:

Preferred:

```
where COLUMN_FILTER[outfitting_order~,~outfitting_order_key~,~oo]
```

Less preferable:

```
where COLUMN_FILTER[ outfitting_order ~,~ outfitting_order_key ~,~ oo]
```

The following tables provide a brief explanation of each SQL macro:

- Table B-2 "Adaptive SQL Macros," on page 231
- Table B-3 "Extraction Set Identification Macros," on page 232
- Table B-4 "Smart Extraction Macros," on page 235
- Table B-5 "Vendor-independent Macros," on page 235
- Table B-6 "Testing Macros," on page 242

SQL Macro Notes

- You can determine the "real" translations for most Epiphany macros by issuing the following SQL in an EpiMeta database:
  Select * from translation_actual

- For usage examples of most of the SQL macros, see the initialization file *templates.sql.*

- The syntax for Epiphany SQL macros may be either
  `$$Macro[arg1~,~ arg2~,~ ...]` or `$$Macro`.

  (Note that ~,~ is used to separate the arguments if there are multiple arguments in the list.)

- If an argument is a list of multiple elements, the elements are separated by commas. For an example, see the macro CREATE_INDEX_IF_NOT EXISTS in Table B-5.

- In some of the macro examples, column spaces have been added near the brackets and near the ~,~ entries for better formatting. In reality, any spaces that are present would be forwarded into the final SQL, so the best coding practice is to avoid them.

- The Usage column in the following tables stands for expected frequency of use.

Table B-2: Adaptive SQL Macros

Macro	Usage	Description
CURR	High	Expands the _A or _B suffix of the currently active tables. Allows you to reference the active or new EpiMart tables.
NEXT	High	Expands the _A or _B suffix of the *not* currently active tables.

Table B-3: Extraction Set Identification Macros

Macro	Usage	Description
COLUMN_FILTER [table_name -,- column_name -,- alias_name]	High	Expands to a "one-sided" SQL comparison expression that requires that alias_name.column_name be greater than or equal to the value in table *table_name* column *column_name* as of the start of the last run. This extracts "everything since last time." Although the alias name is optional: you should use it.
COLUMN_LAST_VALUE [tbl -,- col]	High	Expands to the value of this column as of the start of the last run. This expands to a value, not to an expression, allowing you to make your own expressions.
COLUMN_RANGE_FILTER [table_name -,- column_name -,- alias_name]	High	Expands to a "one-sided" SQL comparison expression that requires alias_name.column_name to be greater than or equal to the value in table *table_name* column *column_name* as of the start of the last run, and less than or equal to this value as of the start of the current run. This extracts "everything since last time, but not including that data that changes while the extractions are running."
DATE_FILTER[column_name]	High	Expands to a "one-sided" SQL comparison expression that requires the column to be greater than or equal to the "current date/ time" as of the start of the last run. This extracts "everything since last time." No table or alias arguments are available to the macro. If the column needs to be qualified, just add the qualifications in the first argument. For example: and $$DATE_FILTER[oo.date_key]

Table B-3: Extraction Set Identification Macros

DATE_RANGE_FILTER [column_name]	High	Expands to a "two-sided" SQL comparison expression that requires the column to be greater than or equal to the "current date/ time" as of the start of the last run, and less than or equal to the current time as of the start of the current run. This extracts "everything since last time, but not including that data that changes while the extractions are running." This compares SQLServer datetimes. The column used for the comparison must be declared as a datetime or some variant in SQLServer. Only the date portion of the value is used: the time portion is discarded.
DATE_LAST_VALUE	High	Expands to the "current date" as of the start of the last run. This expands to a value, not to an expression, allowing you to make your own expressions. This compares SQLServer datetimes. The column used for the comparison must be declared as a datetime or some variant in SQLServer. Only the date portion of the value is used: the time portion is discarded.
TIMESTAMP_FILTER [column_name]	High	Expands to a "one-sided" SQL comparison expression that requires the column to be greater than or equal to the current timestamp as of the start of the last run. This extracts "everything since last time." This compares SQLServer timestamps, which actually have no time or date information in them. The column used for the comparison must be declared as a timestamp type in SQLServer, not as a time or date.

Table B-3: Extraction Set Identification Macros

TIMESTAMP_FILTER_RANGE [column_name]	High	Expands to a "two-sided" SQL comparison expression that requires the column to be greater than or equal to the current timestamp as of the start of the last run, and less than or equal to the current time as of the start of the current run. This extracts "everything since last time, but not including that data that changes while the extractions are running." This compares SQLServer timestamps, which actually have no time or date information in them. The column used for the comparison must be declared as a timestamp type in SQLServer, not as a time or date.
TIMESTAMP_LAST_VALUE	High	Expands to the "current timestamp" as of the start of the last run. This expands to a value, not to an expression, allowing you to make your own expressions.
YYYYMMDD_FILTER [column_name]	High	This is the same as a DATE_FILTER except that only the "day" portion of the date/times is used. (Some business semantics are most meaningful when applied to "days.") This extracts "everything since the last day, including the last day."
YYYYMMDD_FILTER_RANGE [column_name]	High	This is the same as a DATE_FILTER _RANGE except that only the "day" portion of the date/times is used. (Some business semantics are most meaningful when applied to "days.") This extracts "everything since the last day, including the last day, but not including today."
YYYYMMDD_LAST_VALUE	High	Expands to the "current date in YYYYMMDD format" as of the start of the last run. This expands to a value, not to an expression, allowing you to make your own expressions.

Table B-4: Smart Extraction Macros

Macro	Usage	Description
METADBNAME	Medium	Returns the name of the metadata database (EpiMeta).
MARTDBNAME	Medium	Returns the name of the datamart database (EpiMart).
INITIAL_LOAD	Medium	Expands to its argument if the initial load flag is set, or else expands to nothing.
NOT_INITIAL_LOAD	Medium	Expands to its argument if the initial load flag is set, or else expands to nothing.

Table B-5: Vendor-independent Macros

Macro	Usage	Description
ADD_MONTHS [date_expression ~,~ number]	Medium	Takes two arguments: adds the second as a number of months to the first argument, which is a date.
ASSERT_INDEX_EXISTS [index_name]	Low	Causes a SQL error if the index named in argument 1 does not exist. $$BEGIN_ASSERT_INDEX $$ASSERT_INDEX_EXISTS['XPKCustomerMap_B'] $$ASSERT_INDEX_EXISTS['XPKApplicationMap_B'] $$END_ASSERT_INDEX
BEGIN_ASSERT_INDEX	Low	In Oracle, declares the DECLARE INDEX_NOT_EXISTS exception. See ASSERT_INDEX_EXISTS.
CAT	High	Used as an operator to append "two" $$CAT 'strings'. $$TO_CHAR[table1.col1] $$CAT '-' $$CAT $$TO_CHAR[col2]
CHAR_1	Medium	Expands into a type definition for a single character field.

Table B-5: Vendor-independent Macros

CREATE_INDEX_IF_ NOT_EXISTS [index_type~,~ index_name~,~ table_name~,~ column_list~,~ after_creation_clause]	Low	Creates an index if it is not already there. $$DDL_BEGIN $$CREATE_INDEX_IF_NOT_EXISTS[UNIQUE ~,~ XPK_123 ~,~ table1 ~,~ ss_key . iss . date_key . transtype_key . seq ~,~] $$DDL_END

DBNOW	High	Returns the date/time from the database engine.
		
		Select
		  Col1 ss_key
		  $$DBNOW date_modified
		from zork

DDL_BEGIN	Low	Starts a block of code that changes the schema. Use when there is no DECLARE statement already started. $$DDL_BEGIN $$NOT_DEBUG[$$DROP_TABLE_IF_EXISTS[table1]] $$DDL_END
DDL_BEGIN_NO_ DECLARE	Low	Starts a block of code that changes the schema. Use when there is a DECLARE statement already started. DECLARE $$VAR[transFIXED] $$VARCHAR_50$$EOS $$DDL_BEGIN_NO_DECLARE $$VAR_ASSIGN_BEGIN[transFIXED] SELECT $$TO_CHAR[transtype_key] $$VAR_ASSIGN_INTO[transFIXED] FROM Transtype_O WHERE name = 'FINV_ADJUST' $$VAR_ASSIGN_END
DDL_END	Low	Ends a block of SQL that changes the schema. $$DDL_BEGIN $$DROP_TABLE_IF_EXISTS[$$FCTTBL[ ]$$NEXT] $$DROP_TABLE_IF_EXISTS[$$FCTTBL[ ]_INC] $$DDL_END

Table B-5: Vendor-independent Macros

DDL_EXEC [statement]	Low	All items in the argument list are evaluated at runtime. not when the statement is parsed. This macro can construct SQL based on the values of variables computed in the same SQL block. $$DDL_BEGIN $$DDL_EXEC[CREATE INDEX X_table1 ON table1 (iss, ss_key, date_key)] $$DDL_END
DECLARE_BEGIN	Low	Starts a DECLARE block. $$DECLARE_BEGIN $$DECLARE_BODY[$$VAR[count_INC] $$EPIINT] $$DECLARE_BODY[$$VAR[count_FC] $$EPIINT] BEGIN $$VAR_ASSIGN_BEGIN[count_INC] SELECT COUNT(1) $$VAR_ASSIGN_INTO[count_INC] FROM $$FCTTBL[ ]_INC $$VAR_ASSIGN_END
DECLARE_BODY [argument]	Low	Treats its argument as a declaration. See DECLARE_BEGIN.
DROP_INDEX [table_name~,~ index_name]	Medium	Drops the index. $$DDL_BEGIN $$DROP_INDEX[table1 ~,~ index_name] $$DDL_END
DROP_TABLE_IF_EXISTS [table_name]	Medium	Drops the table without returning an error indicating that the table does not exist. $$DDL_BEGIN $$DROP_TABLE_IF_EXISTS[table1] $$DROP_TABLE_IF_EXISTS[table2] $$DDL_END

Table B-5: Vendor-independent Macros

ELSE	Medium	The start of the negative clause of an IF statement.
END_ASSERT_INDEX	Low	Ends a block of checks that indexes exist. See ASSERT_INDEX.
END_IF	Medium	Ends an IF statement. see IF.
EOS	Low	Ends a SQL statement. SELECT 'PROCESSED'. COUNT(1). 1100 FROM table1 $$EOS
EPIINT	Medium	Declares an int. $$DECLARE_BEGIN $$DECLARE_BODY[$$VAR[unjoined] $$EPIINT] $$DECLARE_BODY[$$VAR[processed] $$EPIINT]
EPIKEY	Medium	Declares an Epiphany dimension key. See EPIINT.
FACTMONEY	Medium	Declares a monetary value. See EPIINT.
FACTQTY	Medium	Declares a decimal value. See EPIINT.
FLOAT	Medium	Declares a float value. See EPIINT.
IDENTITY	Medium	Declares a integer serial sequence. See EPIINT.
IF [condition]	Medium	Performs a conditional action. $$IF[$$VAR[fc_exists] = 0] $$DDL_EXEC[$$SELECT_INTO_BEGIN[temp_table] SELECT * $$SELECT_INTO_BODY[temp_table] FROM Old_table WHERE 1=0] $$END_IF $$DDL_END

Table B-5: Vendor-independent Macros

IJ_FROM [table_name1 ~,~ table_name2]	Low	Performs an inner join on two tables. See JOIN_WHERE.
JOIN_WHERE [join_condition]	Low	Supplies the WHERE clause for a join. SELECT col1, col2 FROM Table1 s $$LOJ_FROM[table2 m ~,~ s.iss = m.iss AND s.col2=m.col2] $$LOJ_FROM[table3 d ~,~ m.col1 = d.col1] WHERE 1=1 $$JOIN_WHERE[m.col1=d.col1(+)] $$JOIN_WHERE[s.iss = m.iss (+) AND s.col2 = m.col2 (+)]
LOJ_FROM [join_condition]	Low	Performs a left outer join. See JOIN_WHERE.
MAX_SYS_DATE	Medium	Returns the highest date supported by the database.
NO_FROM_LIST	Medium	Supplies the "dummy" FROM clause needed by some database vendors. SELECT 'MODIFIED'. $$VAR[modified]. 1050 $$NO_FROM_LIST$$EOS
NUMBER(9)	Medium	Declares a decimal(9). See EPIINT.
NUMBER(9,2)	Medium	Declares a decimal(9.2). See EPIINT.
NUMBER(9,5)	Medium	Declares a decimal(9.5). See EPIINT.
NVL [expression ~,~ value]	High	When the first argument is NULL. replace it with the value in the second argument. SELECT $$TO_CHAR[$$NVL[MAX(col1) ~.~ 1]]
ORACLE [expression]	High	Expands to nothing if the database is not Oracle. SELECT COUNT(1) FROM $$SQLSERVER[sysobjects]$$ORACLE[tabs]

Table B-5: Vendor-independent Macros

SELECT_INTO_BEGIN [table_name]	High	Creates a table from a SELECT statement. Expands into a CREATE TABLE AS or a SELECT INTO statement. $$SELECT_INTO_BEGIN[temp_tab] SELECT ■ $$SELECT_INTO_BODY[temp_tab] FROM Old_tab WHERE 1=0
SELECT_INTO_BODY [table_name]	High	Creates a table from a SELECT statement. Expands into a CREATE TABLE AS or a SELECT INTO statement. See SELECT_INTO_BEGIN.
SMALLDATE	Medium	Declares a SMALLDATETIME. See EPIINT.
SMALLINT	Medium	Declares a double-byte integer. See EPIINT.
SQLSERVER [expression]	High	Expands into nothing if the database engine is not SQLServer. SELECT COUNT(1) FROM $$SQLSERVER[sysobjects]$$ORACLE[tabs]
SSKEY	Low	Declares the type Epiphany uses for *ss_keys*. See EPIINT.
SUBSTRING [expression ~,~ start ~,~ length]	Medium	Performs a substring operation. For example: $$SUBSTRING[name ~,~1 ~,~8]
TABLE_EXISTS_ CONDITION [table_name]	Low	Detects if a table exists. SELECT COUNT(1) FROM $$SQLSERVER[sysobjects]$$ORACLE[tabs] WHERE $$TABLE_EXISTS_CONDITION[table_name]

Table B-5: Vendor-independent Macros

TABLE_WITH_PREFIX [database_name -,- table_name]	Medium	Qualifies a table name with a database or user name. SELECT source_system_key iss FROM $$TABLE_WITH_PREFIX[$$METADBNAME -.- source_system]
TINYINT	Medium	Declares a single-byte integer. See EPIINT.
TO_CHAR [expression]	High	Converts a value to a character. Length is second argument. SELECT $$TO_CHAR[$$NVL[MAX(col1) -.- 1]]
TO_DATE [expression]	medium	Converts a value to a database date.
TO_EPIDATE [expression]	High	Converts a date to the string format preferred by EpiChannel. Select col1 ss_key, $$TO_EPIDATE[date_col] date_modified from zork
TO_YYYYMMDD [expression]	Medium	Converts a data to a YYYYMMDD string.
VAR [variable_name]	Medium	References a database variable. SELECT 'PROCESSED'. $$VAR[processed]. 1100 $$NO_FROM_LIST$$EOS
VAR_ASSIGN_BEGIN [variable_name]	Medium	Assigns to a database variable. $$VAR_ASSIGN_BEGIN[max_key] SELECT $$TO_CHAR[$$NVL[MAX(col1) -.- 1]] $$VAR_ASSIGN_INTO[max_key] FROM table2 $$VAR_ASSIGN_END.
VAR_ASSIGN_END	Medium	Assigns to a database variable. See VAR_ASSIGN_BEGIN.
VAR_ASSIGN_INTO [variable_name]	Medium	Assigns to a database variable. See VAR_ASSIGN_BEGIN.

Table B-5: Vendor-independent Macros

VARCHAR_100	Medium	Declares a variable-width character datatype that holds a maximum of 100 characters. See EPIINT.
VARCHAR_15	Medium	Declares a variable-width character datatype that holds a maximum of 15 characters. See EPIINT.
VARCHAR_25	Medium	Declares a variable-width character datatype that holds a maximum of 25 characters. See EPIINT.
VARCHAR_255	Medium	Declares a variable-width character datatype that holds a maximum of 255characters. See EPIINT.
VARCHAR_5	Medium	Declares a variable-width character datatype that holds a maximum of 5 characters. See EPIINT.
VARCHAR_50	Medium	Declares a variable-width character datatype that holds a maximum of 50 characters.See EPIINT.

Table B-6: Testing Macros

Macro	**Usage**	**Description**
DEBUG	Low	Expands to nothing if the **extract** command's verbosity level is not less than 3, otherwise returns its argument.
NOT_DEBUG	Low	Expands to nothing if the **extract** command's verbosity level is not less than 3, otherwise returns its argument

---

EpiCenter Configuration

The Configuration data that appears in the EpiCenter Manager's Configuration dialog box has been set for a default EpiCenter. Other than entering your e-mail password, the information should be correct, or require minimal alteration.

To modify Configuration settings:

Choose Configuration from the EpiCenter menu. The Configuration dialog box (see Figure C-1) is displayed. It has three tabs: Configuration, Transaction Type, and Measure Units. *[The Measure Units tab is not supported for this release.]*

Configuration

A description of the Configuration fields is given in the Configuration dialog box. You may modify these settings if necessary. Select the key in the list and enter a new value in the Value textbox, and click Update.

Note the following:

* You need to change the mail password after installing the Epiphany software.

- current_datamart A / B

  Indicates which set of tables (A or B) is active. See *Mirroring: A and B Tables* on page 55

- date_type

  Calendar is the default.

  date_445 represents the 13-week-per-quarter calendar in which the months in the quarter are defined as consisting of 4 weeks, 4 weeks, and 5 weeks.

- fiscal_year start_m[onth]

  If the chosen month is other than January, then the actual starting year is one less than the start_year value (thus the given year may start on January 1 and still be a complete fiscal year).

- number_of_years

  The number of years during which the data warehouse will be operational. The default is 20.

- start_day_445

  The beginning day of the 445 quarter.

- start_year

  The year that the EpiMart becomes operational. The default is 1990.

- version

  The version number of this EpiCenter Manager.

- week_start_day

  The day of the week that is the first day of the week. Sunday is the default.

- last_extract_date

  The date that appears as the *Data is valid as of...* in Clarity reports.

Figure C-1 Configuration Tab

Transaction Types

The current transaction types are shown in the Transaction Types tab (Figure C-2) of your Configuration dialog box. These are the typical transaction types for a generic installation. Your site may need additional transaction types for complicated measures.

Use this tab to customize the transaction types for your site. Keys 1-99 are reserved for booking transaction types. Keys 101-199 are reserved for shipping transaction types.

After modifying the transaction types, click Update, which writes the new data to the EpiMeta.

Figure C-2 Transaction Type

Measure Units

[The Measure Units tab is not supported for this release.]

Measure units define how currency units, percentages, and numerical units are displayed by default on ticksheets. Enter the name of the unit and a description in the textboxes in the Measure Units tab and click Add Unit. The system generates a hidden key for the unit. Web Builder uses this key when flagging a measure with a unit designation.
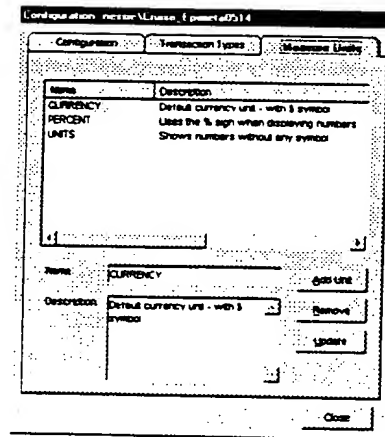
Figure C-3 Measure Units Tab

APPENDIX D

Date Dimension Fields

The date dimension fields used by the Epiphany system are described below.

Table D-1: Date Dimension Fields

dim_col_name	Description
cq_and_cy_name	Calendar quarter and year name. For example, Q1 1998.
cq_name	Calendar quarter name. For example, Q1.
cy_name	Calendar year name. For example, 1998.
date_key	Primary key—date as a native date type.
day_cq_begin	Whether or not this is a day on which a calendar quarter begins. (1/0).
day_cq_end	Whether or not this is a day on which a calendar quarter ends. (1/0).
day_cy_begin	Whether or not this is a day on which a calendar year begins. (1/0).
day_cy_end	Whether or not this is a day on which a calendar year ends. (1/0).

Table D-1: Date Dimension Fields

dim_col_name	Description
day_fq_begin	Whether or not this is a day on which a fiscal quarter begins. (1/0).
day_fq_end	Whether or not this is a day on which a fiscal quarter ends. (1/0).
day_fy_begin	Whether or not this is a day on which a fiscal year begins. (1/0).
day_fy_end	Whether or not this is a day on which a fiscal year ends. (1/0).
day_month_begin	Whether or not this is a day on which a month or period begins. (1/0).
day_month_end	Whether or not this is a day on which a month or period ends. (1/0).
day_name	The day as a native date type.
day_name_char	Date (as a string). For example, Apr 02 1995.
day_name_char_weekday	Date (as a string) with weekday prefix. For example, Sun Apr 02 1995.
day_number_in_cq	The number of this day in the calendar quarter, starting at 1.
day_number_in_cy	The number of this day in the calendar year, starting at 1.
day_number_in_fq	The number of this day in the fiscal quarter, starting at 1.
day_number_in_fy	The number of this day in the fiscal year, starting at 1.

Table D-1: Date Dimension Fields

dim_col_name	Description
day_number_in_month	The number of this day in the month or period, starting at 1.
day_number_in_week	The number of this day in the week, starting at 1.
day_number_til_end_cq	The number of days until the end of the calendar quarter, ending with 1.
day_number_til_end_fq	The number of days until the end of the fiscal quarter, ending with 1.
fq_and_fy_name	Fiscal quarter and year name. For example, Q1 1998.
fq_name	Fiscal quarter name. For example, Q1.
fy_name	Fiscal year name. For example, 1998.
month_and_cy_name	Month (abbrev.) name and calendar year. For example, Apr 1998.
month_and_fy_name	Month (abbrev.) or period name and fiscal year. For example, Apr 1998.
month_number	Month or period number since the first date in the system, starting at 1.
month_number_in_cq	Month number since the start of the calendar quarter, starting at 1.
month_number_in_cy	Month number since the start of the calendar year, starting at 1.
month_number_in_fq	Month or period number since the start of the fiscal quarter, starting at 1.

Table D-1: Date Dimension Fields

dim_col_name	Description
month_number_in_fy	Month or period number since the start of the fiscal year, starting at 1.
month_number_til_end_cy	Number of months or periods until the end of the calendar year, ending with 1.
month_number_til_end_fy	Number of months or periods until the end of the fiscal year, ending with 1.
vacation_day	Whether or not this day is a vacation, (1/0).
week_friday	Date (as a string) for the Friday of the current week. For example, Apr 01 1994.
week_monday	Date (as a string) for the Monday of the current week. For example, Apr 01 1996.
week_number	Week number since the first date in the system, starting at 1.
week_number_cq	Week number since the start of the calendar quarter, starting at 1.
week_number_cy	Week number since the start of the calendar year, starting at 1.
week_number_fq	Week number since the start of the fiscal quarter, starting at 1.
week_number_fy	Week number since the start of the fiscal year, starting at 1.
week_number_til_end_cq	Week number until the end of the calendar quarter, ending with 1.
week_number_til_end_cy	Number of weeks until the end of the calendar year, ending with 1.

Table D-1: Date Dimension Fields

dim_col_name	Description
week_number_til_end_fq	Week number until the end of the fiscal quarter, ending with 1.
week_number_til_end_fy	Number of weeks until the end of the fiscal year, ending with 1.
week_saturday	Date (as a string) for the Saturday of the current week. For example, Apr 01 1995.
week_sunday	Date (as a string) for the Sunday of the current week. For example, Apr 02 1995.
weekday	Weekday prefix. For example, Sun.

Date Dimension Fields

APPENDIX E

Physical Type Values

Table E-1, *Oracle Physical Type Values*, and Table E-2, *SQL Server Physical Type Values*, define the database type translations for the physical types you select in EpiCenter Manager's Base Dimension and External Tables dialog boxes. The physical type you select is replaced by its associated database type.

The translation of physical type to database type depends on your RDBMS platform.

Table E-1: Oracle Physical Type Values

Physical Type	Database Type
CHAR_1	CHAR(1)
EPIINT	NUMBER(9)
EPIKEY	NUMBER(9)
FACTMONEY	NUMBER(16,4)
FACTQTY	NUMBER(16,4)
FLOAT	FLOAT
IDENTITY	NUMBER(9)
NUMBER(9)	NUMBER(9)

Table E-1: Oracle Physical Type Values

Physical Type	Database Type
NUMBER(9.2)	NUMBER(9.2)
NUMBER(9.5)	NUMBER(9.5)
SMALLDATE	DATE
SMALLINT	SMALLINT
SSKEY	VARCHAR2(50)
TINYINT	SMALLINT
VARCHAR_5	VARCHAR2(5)
VARCHAR_15	VARCHAR2(15)
VARCHAR_25	VARCHAR2(25)
VARCHAR_50	VARCHAR2(50)
VARCHAR_100	VARCHAR2(100)
VARCHAR_255	VARCHAR2(255)

Table E-2: SQLServer Physical Type Values

Physical Type	Database Type
CHAR_1	CHAR(1)
EPIINT	INT
EPIKEY	INT
FACTMONEY	MONEY
FACTQTY	DECIMAL(16,4)
FLOAT	FLOAT
IDENTITY	INT IDENTITY
NUMBER(9)	DECIMAL(9)
NUMBER(9,2)	DECIMAL(9,2)
NUMBER(9,5)	DECIMAL(9,5)
SMALLDATE	SMALLDATETIME
SMALLINT	TINYINT
SSKEY	VARCHAR(50)
TINYINT	SMALLINT
VARCHAR_5	VARCHAR(5)
VARCHAR_15	VARCHAR(15)
VARCHAR_25	VARCHAR(25)
VARCHAR_50	VARCHAR(50)
VARCHAR_100	VARCHAR(100)
VARCHAR_255	VARCHAR(255)

Physical Type Values

APPENDIX F

Writing Staging SQL Statements

The Epiphany extraction process (excluding aggregation) consists of two phases: 1) loading data from source systems into the Epiphany staging tables, and 2) running semantic instances against these staging tables. This appendix describes the recommended practices for writing SQL statements that populate the dimension and fact staging tables.

Base Dimension Staging SQL Statements

Base dimensions describe the physical dimension tables in EpiMart. You can use EpiCenter Manager to configure the dimension columns for each base dimension table. During an extraction job, one or more SQL statements can be executed against the base dimension staging table to populate these columns.

The purpose of the base dimension staging query is to extract the latest values from the source system. You need not be concerned with determining when a value has changed because Epiphany's semantic templates perform this task.

You can use the Template feature in the SQL Statement dialog box (see Figure 3-19, page 98) to provide an SQL template for a given dimension table. For example, assume you define a base dimension called Customer with the following dimension columns:

- FullName

- Age

- City

- State

- ZipCode

In the Tables References panel of the SQL Statement dialog box, select the option *Populates a dimension.* Choose the dimension table (Customer) from the drop-down list. Clicking the Template button will display the following SQL in the dialog box:

```
SELECT
    <YOUR EXPRESSION> customer_sskey,
    <YOUR EXPRESSION> data_modified,
    <YOUR EXPRESSION> FullName,
    <YOUR EXPRESSION> Age,
    <YOUR EXPRESSION> City,
    <YOUR EXPRESSION> State,
    <YOUR EXPRESSION> ZipCode
FROM
    <YOUR TABLE>
```

This is a regular SQL statement for which you must provide the values <YOUR EXPRESSION> and <YOUR TABLE>. You must assign each column in the SELECT list a valid SQL expression for the value of its destination column. A FROM clause is required to make this a valid statement; a WHERE clause is optional.

The first two columns, *customer_sskey* and *date_modified*, were not specified in the definition of the Customer base dimension. These columns are implicitly added to the base dimension table by Epiphany's Adaptive Schema Generator and must be populated by the staging SQL. The first column is a source system key (*sskey*) and should be unique for every row in the staging table. The concept of *sskey* is important in EpiMart because it tells the semantic templates whenever a row in the staging table represents the same source system entity as a row that already exists in the dimension table. The *sskey* is a variable length string, normally of maximum length 50. It corresponds to the primary key of the source system table or the tables that make up this query.

> *Note:* If the *sskey* column is of interest to end users for querying, then a dimension column populated with the same values will need to be created because the *sskey* is not available for ticksheet configuration.

The other implicit column, *date_modified*, has the same name in all base dimension staging tables and is used to identify when a base dimension row is inserted into the EpiMart. If the source system contains a *creation date* field, then this field should be used. Otherwise, you can use the source system's expression for "right now," which causes newly extracted rows to assume the date when they were extracted into EpiMart; for Microsoft SQLServer this expression is GetDate(). For best results, dates should be returned as strings, for example, 5/1/1998.

The remainder of the columns in the SELECT list must be populated with an appropriate expression for the meaning of that column. Any SQL expression that can be executed against the source RDBMS is valid. Also, Epiphany provides a set of SQL macros that will be automatically expanded to the correct syntax for your source system. Use of macros facilitates the cross platform usage of your SQL Statements. See Appendix B, *Epiphany Macros*, for more information.

An important point about these expressions is that null values are *not* allowed in any field of the staging table. The reason for this is simple: the Epiphany system uses GROUP BY statements at end-user query time to form the tables and charts of front-end applications such as Clarity. However, fact rows that aggregate on columns with null values are left out of the resulting reports because nulls are removed from GROUP BY's. Rather than incurring the query-

time penalty for this check, EpiMart insists on non-null dimension column values.

To circumvent this problem, substitute the string UNKNOWN for any null values using the NVL macro. The Epiphany system will automatically generate an UNKNOWN row in your dimension table. The UNKNOWN value is configurable; if UNKNOWN is a valid value in your dimension table, use another value.

Duplicate sskey's

If during a single extraction, a staging table is loaded with two or more rows with the same *sskey*, then the last row entered is used. See Appendix G for a description of the dimension semantic types.

Dimension Staging Queries with Joins

The Epiphany system allows the use of joins in base dimension staging queries. Star schemas typically de-normalize data structures in transactional systems into flat hierarchies, and you must be aware of what the granularity of a base dimension represents in this circumstance.

For example, you will rarely want to use a Cartesian product of two tables in a base dimension staging query, unless the *sskey* of the result set will combine the primary keys of the two tables that are being crossed. It is more common for a single table to "drive" the result set, with other tables joined through unique key lookups to provide additional textual values. For instance, a Product Master table in the source system might represent the driving table of a Product base dimension (with the *sskey* taken from the primary key of the Product Master table), but other tables with textual values for Product Line or Platform may be joined with this master table. In this case, you should ensure that the joined columns of the lookup tables are properly indexed (usually with UNIQUE indexes).

Constructing Base Dimension Queries with DISTINCT Fact Values

Sometimes dimensions are created for which no corresponding master table exists in the source system. For instance, an Order fact may have an Order type associated with it (with several possible choices). These values will be embedded directly in the fact rows on the source, but no lookup table exists with all the choices. In this case, a SELECT DISTINCT query against the source system's fact table might be appropriate for populating base dimension staging tables in EpiMart. The alternative to this method is the use of degenerate dimensions in the fact table, although degenerate dimensions cannot be aggregated.

Fact Staging SQL Statements

SQL statements that populate fact staging tables are generally more complex than the ones used to load dimension staging tables. As with base dimension tables, the columns of the SELECT statements are determined by the metadata definition of the fact table (and its constellation) along with certain implicit rules.

To illustrate this point, assume that you define an Order fact in a Sales constellation with the following dimension roles:

- CustomerBillTo
- Product
- CustomerShipTo
- SalesPerson

The constellation contains a single degenerate dimension called OrderNumber, and the Order table has two fact columns: *net_price* and *number_units* that represent the extended amount for an order line item, along with the quantity.

Clicking the Template button on the SQL Statement dialog box (with the *Populates fact table* option selected and the Order table selected in the drop-down list) displays the following SQL in the dialog box:

```
SELECT
    <YOUR EXPRESSION> ss_key,
    <YOUR EXPRESSION> date_key,
    <YOUR EXPRESSION> transtype_key,
    <YOUR EXPRESSION> process_key,
    <YOUR EXPRESSION> customerbillto_sskey,
    <YOUR EXPRESSION> product_sskey,
    <YOUR EXPRESSION> customershipto_sskey,
    <YOUR EXPRESSION> salesperson_sskey,
    <YOUR EXPRESSION> ordernumber_key,
    <YOUR EXPRESSION> net_price,
    <YOUR EXPRESSION> number_units
FROM
    <YOUR TABLE>
```

As with base dimension staging queries, you must identify what a row in this fact table represents. Based on the columns in this example, a row seems to indicate a line item of a sales order. (In this case, the assumption is that the salesperson gets full credit for a line item; another interpretation of this fact row might be a particular amount of credit that a salesperson received for an order line item.) Typically, the FROM clause of this query would join the Order Line Item table to the Order Header table in the source system.

The columns in the SELECT list can now be divided into these categories:

- Implicit columns that were added automatically

- Dimension role foreign keys

- Degenerate dimension keys

- Fact numeric columns

First, consider implicit columns. As with base dimensions, each fact staging row contains an *ss_key* (notice the difference in spelling) that uniquely identifies this row in the source system. In this example, the *ss_key* might be a concatenation of the Order Number with the Order Line Number (since this combination is presumably unique). *ss_key*'s will be used on subsequent extractions to prevent duplicate copies of the fact row from being created in EpiMart.

The *date_key* indicates when the fact occurred. Since time is a central component of EpiMart, each fact table must contain this column. Many facts are time based; in this example, *date_key* represents the time when the order was placed. However, if time is not important for this fact, then the current system time can be used as a placeholder. Note that *date_key* is granular only to that single *day* when the fact occurred. For best results, the fact SQL Statement should return the day as a string, for instance, 5/1/1998.

Transaction type is another central concept of fact table processing in EpiMart. The SQL statement should return a numeric key that matches with one of the transaction types defined on the Configuration dialog box in EpiCenter Manager. (See Appendix G, *Semantic Types*, for more information about *transtype_key*.)

The *process_key* identifies rows from the fact table to be processed by a specific semantic type. (A fact table can contain different types of rows, requiring different semantic types.) For example, the Order fact might hold both Bookings and Shippings, and *process_key* would identify which fact staging rows were which. (See Appendix G, *Semantic Types*, for more information about *process_key*.)

Next, you must enter values for each of the dimension role foreign keys. Notice that the names of the columns in the SQL template are *DimRoleName_sskey*. These fields refer back to *sskey*'s of the base dimension tables for this fact. You need to understand the meaning of each base dimension table to ensure that the keys resolve properly. If the *sskey* of the Product base dimension is taken from a Product Master list in the source system, then *product_sskey* in the Order table must also refer to an entry in the Product Master. If a base dimension is the cross-product of two source system tables, the fact staging keys for that dimension must also represent a unique cross-product entry.

Degenerate keys in the fact staging query should be populated with string values. In the example above, the *ordernumber_key* field would probably be populated with the actual primary key of the Order Header table, such as Order Number 253AD56.

Finally, the numeric columns represent the actual quantities and raw amounts that are associated with each fact entry. Each column should be an additive amount for correct front-end query results. For instance, total dollar amounts for a line item should be populated instead of unit prices because unit prices cannot be added across fact rows.

Using External Tables as Inputs to Staging Queries

Sometimes it may be necessary to bring data into EpiMart external (temporary) tables before performing any joins; for example, if the source system's SQL limits your ability to manipulate the data. The full power of EpiMart's RDBMS engine can then be used to load the staging tables. In this case, the following sequence of actions is usually employed:

1. Drop any indexes on the external tables for fast loading.

2. Load the external tables from the source system.

3. Create any indexes on external tables needed for fast joins.

4. Load the staging tables using queries against the EpiMart External tables. All query plans should use the indexes built in Step 3.

APPENDIX G

Semantic Types

This appendix describes the dimension and fact semantic types.

   Note: Fact rows with all zero facts are discarded by all semantics.

Dimension Semantic Types

In the initial release of Clarity and Relevance 3.2, there is the only one dimension semantic type: Slowly Changing Dimensions.

The Release 3.2 Service Pack 1 adds these dimension semantic types: Latest Dimension Value, First Dimension Value, and Initial Dimension Value.

Slowly Changing Dimensions

A Slowly Changing Dimension is a dimension in which the attributes or hierarchy of the dimension can change over time, but historical data is not restated. Two examples follow:

- A sporting goods chain decides to rename a store. By using the Slowly Changing Dimension semantic type, the old name is retained for all of the historical data. One can still identify when the store changed names and compare sales before and after the name change.

- This same chain is national and has stores divided into three regions. On January 1, 1998, the chain reorganizes its regions, moving Denver from the Central to the Western region. Their 1998 sales forecasts take into account that the Denver store is in the Western region, but they do not want to recalculate forecasts and actuals from previous years. Using the Slowly Changing Dimension, sales for Denver can be aggregated up to the Central region through 1997. Beginning January 1, 1998, Denver sales are applied to the Western Region.

The Slowly Changing Dimension semantic type accomplishes the following logic:

- Rows with the same *sskey* in the staging table will be eliminated following in a "last in wins" rule. This is determined by the special column called *ikey*, which is created by EpiChannel and is automatically incremented during normal extractions. The highest *ikey* row for a given *sskey* will be accepted.

- New rows are created by searching through the dimension staging table for new *sskey* values, or *sskey* values that have one or more dimension column changes from the last known values. Each of these cases creates a new row in the dimension table. The mapping row for that *sskey* points to the latest dimension row with that *sskey* value.

- In the EpiChannel log file, new *sskey* rows are reported in the *Inserted* column. The number of changed rows is reported in the *Modified* column. The number of rows in the staging table is reported in the *Processed* column.

- The dimension table has its key column made into a Primary Key index. Each dimension column is also Indexed (non-uniquely). The Mapping table is indexed (primary key) on *iss, sskey*.

  iss is used when two source systems have the same *sskey* values (such as when the SAP and Vantive systems both have a Customer #2 record). *iss* is determined by the source system identifier selected using the General tab of the Data Store dialog box (Figure 3-15, page 91).

- The column *dimension_key_REAL* (where *dimension* is the dimension column name) is set to the first key value for each *sskey*. In other words, when a new *sskey* is discovered, then the *REAL* key is set to the new dimension key value. Subsequent dimension rows for this *sskey* will retain the original REAL key value.

- The UNKNOWN *sskey* always maps to dimension key value 1 in the Mapping table.

You need to be aware of the following:

- Do not allow dimension column values to *oscillate* unpredictably. (See *First Dimension Value* on page 270 for more information.) In particular, do not rely on *ikey* filtering of duplicate *sskey* values if two or more rows during a single extraction might have different values for one or more dimension columns. The reason is that a new row will be created in the dimension table for every extraction for which a change is recorded. This can cause two values to "compete" with each other, forcing an unending sequence of row creation in the dimension table.

- The only way to remove rows from dimension tables once they have been extracted is with an explicit delete or truncation.

Latest Dimension Value

(Release 3.2 Service Pack 1)

The Latest Dimension Value semantic type applies changes retroactively to a dimension. Thus the changes take effect for all historical data, as well as for current and future loads.

For example, assume that a sporting goods store has a category historically called *Rollerblades*. Now that they are selling other brands, the store wants to change the category to *In-line Skates*. By using the Latest Dimension Value semantic type, this change can affect all of the historical data because all previous sales of *Rollerblades* are now labeled *In-line Skates*. As a result, the store can compare year-to-year sales of all in-line skates.

This semantic type has the same duplicate *sskey* filtering as Slowly Changing Dimensions. Use this semantic type for an implementation that "restates history" when a source dimension table changes.

Note the following:

- New *sskey*'s are inserted in both the dimension table and mapping table. Existing *sskey*'s with one or more changed dimension columns are updated in place in the dimension table (that is, the same dimension row is used) with the latest values.

- In the EpiChannel log file, new *sskey* rows are reported in the *Inserted* column. The number of changed rows is reported in the *Modified* column. The number of rows in the staging table is reported in the *Processed* column.

- Latest Dimension Value has the same indexing as Slowly Changing Dimensions.

- The REAL column is always equal to the dimension key.

- Latest Dimension Value has the same UNKNOWN mapping behavior as Slowly Changing Dimensions.

First Dimension Value

(Release 3.2 Service Pack 1)

The First Dimension Value semantic type ignores any changes to a dimension. The values that were present when the row was first inserted are preserved forever, regardless of any future changes.

You might want to use this type if your source data comes from two systems that are not in complete agreement with each other. For instance, if one system has Customer #12345 as *David Anderson*, and the other has the same customer as *David Andersen*. Ideally, you would determine which one was in error and correct it.

In the meantime, you could choose to apply the first value read and to ignore the other. (This is a good method for avoiding the *oscillation* problem mentioned on page 269.) If you were to use the Slowly Changing Dimension semantic type in this case, there would be a race between the two source systems for each extraction, and (in the worst case), your dimension values could alternate between the two values with every extraction.

Note the following:

- First Dimension Value has the same duplicate *sskey* filtering as Slowly Changing Dimensions.

- New *sskey*'s are inserted in both the dimension table and the mapping table. Existing *sskey*'s are ignored.

- In the EpiChannel log file, new *sskey* rows are reported in the *Inserted* column. The number of rows in the staging table is reported in the *Processed* column.

- First Dimension Value has the same indexing as Slowly Changing Dimensions.

- The REAL column is always equal to the dimension key.

- First Dimension Value has the same UNKNOWN mapping behavior as Slowly Changing Dimensions.

Initial Dimension Load

(Release 3.2 Service Pack 1)

The initial Dimension Load semantic type is used to load the dimension without regard to any previously existing rows. This can be used for the initial load of the empty EpiMart, and also to completely reload a dimension, ignoring existing values. Using any other semantic type would require emptying the existing dimension table before beginning the extraction.

Note the following:

- Initial Dimension Load has the same duplicate *sskey* filtering as Slowly Changing Dimensions.

- The existing dimension and mapping tables are ignored; all *sskey*'s are imported directly into the dimension and the mapping tables.

- In the EpiChannel log file, new *sskey* rows are reported in the *Inserted* column. The number of rows in the staging table is reported in the *Processed* column.

- Initial Dimension Load has the same indexing as Slowly Changing Dimensions.

- The REAL column is always equal to the dimension key.

- Initial Dimension Load has the same UNKNOWN mapping behavior as Slowly Changing Dimensions.

Fact Semantic Types

Update Unjoined (Optional)

Always run this semantic type *before* any other semantic type for a fact table. Its purpose is to check the referential integrity of all dimension *sskey*'s in the fact staging table and to set all those keys that do not resolve to valid dimension *sskey*'s to the special value UNKNOWN.

The special *sskey* value of UNKNOWN always maps to the dimension key value 1 (the UNKNOWN dimension row). By setting all unresolved dimension *sskey*'s to UNKNOWN, those fact entries will be mapped to this special dimension row. All reports run against these fact rows will yield the default values for that dimension (see *The UNKNOWN Dimension Row* on page 34 and *The Update Unjoined Semantic Type* on page 36 for more information).

Update Unjoined uses the dimension mapping tables to determine whether a given dimension *sskey* is valid. For this reason, all dimension semantic instances must be executed before any fact semantic type.

In the EpiChannel log file, the number of changed rows is reported in the *Modified* column.

Custom Fact Index

In EpiCenter Manager, you use the Custom Index tab of the Fact Table dialog box to define custom indexes for a fact table. The semantic type Custom Fact Index, however, must be executed for the fact table as part of the scheduled extraction process in order for these indexes to be built.

> *Important.* Schedule this semantic Instance *after* all other semantic instances for a fact table.

For a discussion of custom indexing, see *Custom Fact Indexing* on page 33.

Transactional

The Transactional Fact Semantic Type is the simplest means of moving fact staging data into fact base tables. This semantic type uses the following logic:

- Only rows with *process_key* set to 1 (Transactional) are used; others are discarded.

- For *sskey*'s that are already entered in the current fact base table, all rows in the staging table with that *sskey* and with the same or earlier *date_key* are discarded. If the *sskey* already exists in the fact table, only later dates can be added to the fact table.

  Two or more rows with the same *sskey* can be imported into the fact base table if they arrive in the same extraction and the *sskey* either does not already exist, or exists but is dated earlier. (This property is used by the *pseudo-order* approach to the Booking/Shipping problem; see *Transactional/State-like/Force Close* on page 275.)

- In the EpiChannel log file, all rows added to the fact base table are reported in the *Inserted* column, and the total number of rows in the staging table is reported in the *Processed* column. Any rows whose dimension *sskey*'s do not resolve to valid values are reported as Unjoined. (This occurs only if Update Unjoined is not used; see *Update Unjoined (Optional)* on page 272.)

Note the following:

- Transactional semantics should be used for event facts; once the fact event has occurred, it can never be modified.

- To reload transactions after they have been loaded into the fact base table, the rows must be deleted from the fact base table, or the fact base table must be truncated.

Transactional/State-like

The Transactional/State-like semantic type allows changes to already existing rows in EpiMart. It uses the same logic as the Transactional semantic type, with the addition of the logic described below.

First these three steps occur:

1. Records in the staging table with *process_key* of 2 (state-like) are treated as Orders that can be *differenced* between extractions (a discussion of this follows).

2. For records in which *process_key* = 2, duplicate *sskey*'s with the same *date_key* in the same extraction cause only the highest *ikey* value to be used. Other rows are discarded. This is the same filtering that happens in dimension semantics such as Slowly Changing Dimensions.

3. For an *sskey* with the highest *ikey* (which means that for every set of rows with the same *sskey*, take the row with the highest *ikey*), if the *date_key* for that staging row is less than the last *date_key* for that *sskey* in the fact base table, the staging row is discarded. In other words, an Order can only be modified on its last reported date, or some time further into the future.

After Steps 1-3, the staging fact columns and dimension values are compared with the current values in the fact table (if any). Adjusting records are created in the fact table so that the fact table now reflects the reported *state* from the staging table. (This is why the term *state-like* is used.) *Differenced* transactions are invented if the numeric fact columns have changed.

If the dimensionality has changed, then the Order will be "de-booked" and "re-booked" with the correct dimensionality.

If the same *sskey* appears in the staging table with more than one *date_key*, then further adjusting transactions are made in the fact base table as appropriate to bring the fact base table in line with the reported staging rows.

> *Note*: By convention, Bookings are entered with positive facts (negative for Returned Orders), whereas Shipments are entered with negative facts (positive for Returned shipments).

When using this semantic type it is difficult to ensure that Backlog calculations will remain consistent when Orders are not completely closed in the source system. Use the Transactional/State-like/Force Close instead.

Transactional/State-like/Force Close

This semantic type is equivalent to Transactional/State-like with the following additional logic.

Once a Booked Order is entered into EpiMart, it remains in that state (with an Open Backlog) forever. In normal scenarios, an invoice eventually arrives, which will close the Backlog. However, in some source systems, Booked Orders can be removed from the system completely. If such an Order had been entered previously into EpiMart, then it will remain in an Open Backlog condition forever.

The solution to this problem is to use Transactional/State-like/Force Close, which establishes a "Challenge Protocol" for Open Orders. In this scenario, all Open Bookings must be extracted into the staging table during every extraction. The reason is that the Force Close logic will close out all Open Orders (*sskey*'s with non-zero facts in the system) that do not appear in the fact staging table. Only Booking Transaction types (those with *transtype_key* values between 1 and 99) will be affected in this manner.

When using this semantic type, the methodology that has been found to work in practice involves the use of pseudo-orders as follows:

- One extraction SQL statement extracts all Open Orders. Fact amounts are the Open amounts (what has not been shipped yet), not the ordered amounts. Transaction types for this statement should be in the Booking range (1...99), and *process_key* should be 2 (state-like).

- The second extraction statement uses *process_key* of 1 (Transactional) and represents all shipments in the system. These records normally go into the system with negative facts for positive shipments (by convention). These transaction types should be in the shipment range (101 or greater).

- The third statement is a restatement of the shipment, but as a Booking (*transtype_key* between 1...99). The *process_key* is still 1 (allowing the Transactional Semantics to import it), but the transaction type is a Booking. The same *sskey* and dimensionality as the second statement are used. These are the pseudo-orders since they are actual shipments that are entered as Orders.

The net effect of this methodology is that as a shipment is reported against Bookings, the Open Booking quantity in statement 1 is decremented, while the actual shipment is restated as a Booking transaction. Eventually, when the Order is removed from the system, the Force Close logic will close out any remaining Open fact quantities from statement 1 above. What remains in the system will be Shipments and their corresponding pseudo-orders. By construction, the Backlog will be zero.

Transactional/Inventory

Not available for this release.

Transactional/Inventory/Force-zero

Not available for this release.

Pipelined

Use the Pipelined semantic type for facts that can exist in several different life-cycle phases, called pipeline states; for example, sales opportunities or support calls facts.

The fact staging table contains an extra column called *pipe_state*, which serves the special purpose of tracking the position of a given *sskey* in the pipeline of this fact. The designer of this fact table should organize a numbering system for this column in which a larger *pipe_state* number means forward movement in the pipeline. In order to report on this special column, a dimension should be used which tracks changes to the *pipe_state* column (since front-end queries cannot be run against *pipe_state*).

The following logic is used:

- Records with the same *sskey* and *date_key* are filtered using the highest *ikey* value as described above.

- *sskey* records already in the fact base table are filtered in the same manner as for *Transactional/State-like*. That is, only the latest *date_key* or greater for an *sskey* will survive this filter.

- Only records with *process_key* of 3 (Pipelined) are used.

- For the records that pass filters 1-3, appropriate transactions will be created in the fact base table to place each *sskey* into its proper pipeline stage. When an *sskey* first enters a given pipeline stage, a record will be created for the Booking in that stage. The *transtype_key* is taken from

the fact staging table (recommended range of 4...99). When the same *sskey* subsequently moves forward in the pipeline then a SHIP record (*transtype_key* = 101) will be created in the old pipe state, while a Booking will be created in the new pipe state (again with the *transtype_key* supplied in the staging table). Similarly, backwards movement in the pipeline causes a LOST (*transtype_key* = 3) transaction to be inserted along with a corresponding new Booking.

- A change of dimension within a pipeline stage causes an appropriate delta transaction to be created (the *transtype_key* is taken again from the fact staging table).

APPENDIX H

Export/Import of Metadata

All of the control information for an EpiCenter is stored in a single metadata repository called EpiMeta. EpiMeta represents a transactional, fully relational model of over one hundred and fifty tables, with complicated declarative referential integrity constraints. Epiphany provides tools such as Web Builder and EpiCenter Manager for configuring this metadata without the need to write to, or even know about, the underlying data structures.

Epiphany also provides a metadata Export/Import utility for moving metadata between EpiCenters and for backing up the definition of an EpiCenter. To use this tool properly, you should understand the basic metadata concepts discussed in this appendix.

Metadata Overview

All of the user-configurable metadata tables have integer primary keys. These non-natural keys are provided by the database engine when a metadata row is inserted; the value of the primary key itself has no intrinsic meaning. All relationships to this inserted row are made via this integer. For instance, the relationship between a filter group and its filter block is represented in EpiMeta via an integer column called *filter_block_key*.

The Access Export database does not simply contain a copy of the metadata tables being exported for this reason: If data were copied to the Export file. then when this same information is imported into a new EpiMeta. the integer primary key values in the Export file might clash with already existing primary keys in the target EpiMeta. Therefore, the Access database uses an EpiMeta-independent representation of metadata. See *Export File Format* on page 282 for a description of these Access database tables.

EpiMeta contains many tables. all of which are inter-related. Ticksheet metadata refers to constellation metadata (for instance, the attributes refer to dimension columns), while security metadata refers to ticksheets. In order to export only a portion of the metadata at a time, the Export machinery must decide where to stop exporting—otherwise, the entire metadata must be exported with each operation.

When using the Export metadata command of EpiCenter Manager. you must select which part of the metadata to export. Figure H-1 shows an overview of the various domains of metadata within EpiMeta.

Each rectangle in the figure represents one of the options available for export. Everything within the rectangle is actually contained as metadata in the Export file when that option is selected. The arrows in the diagram represent references to other metadata. These references are contained in the Export file as well, but the references are made to other objects by name only. In other words, when exporting ticksheets, the measure mapping metadata is exported in the Export file, but the measures themselves are not exported (unless you also select the Measure option). Only a reference to the appropriate measures is exported.

Upon import. these references are used as follows. When ticksheet metadata is imported into a different EpiMeta, the import machinery searches for measures with the referenced names. If it finds these, then the same relationships are established in the new EpiMeta as the ones that were exported. However, if the Import machinery does not find these measures by name, the measure mapping information will be lost upon import.
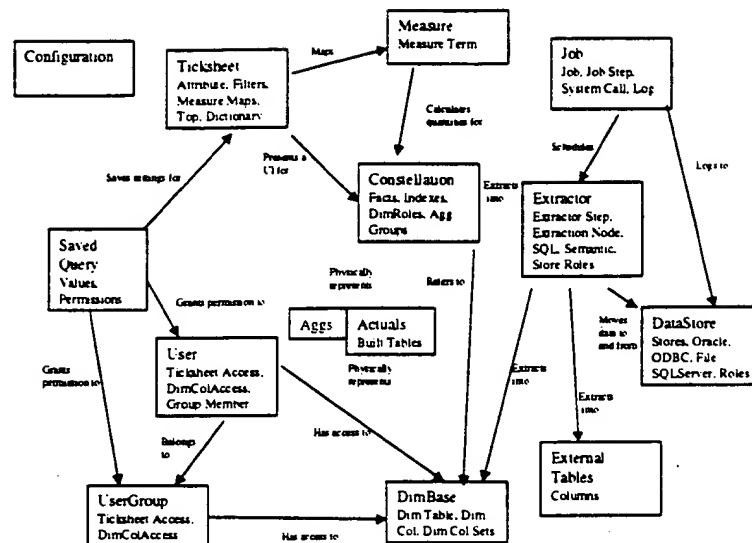
Figure H-1 EpiCenter Data Model

Replacing Existing Metadata on Import

When importing metadata into an existing EpiMeta, the Import machinery will detect an attempt to overwrite existing metadata. The definition of "existing" is usually based on a unique name column for one of the metadata tables. For instance, ticksheets must have unique names, so an attempt to import a ticksheet with the same name as an existing one results in a warning message (unless the Always Replace option has been selected previously).

The Release 3.2 Import machinery uses a Delete/Re-insert model to replace existing metadata. In future versions, a row will be modified "in-place" instead of first being deleted. However, it is important to understand the ramifications of this delete-first operation.

When a row of metadata is deleted, all rows that reference that row are also deleted. For example, suppose a single measure is imported over one with the same name. If ticksheets already refer to that measure, then the Import operation deletes the measure mappings to those ticksheets. First, the measure is deleted (which deletes all the measure mappings to that measure). Then the measure itself is re-imported, but since the Export file does not contain the ticksheet metadata, the measure mappings are not re-imported. Thus exporting metadata and then immediately re-importing it does not produce an equivalent EpiMeta.

Actuals and Agg Metadata

The Export All command in EpiCenter Manager does not export the entire contents of EpiMeta. The options for Actuals and Aggs are omitted by default from this export. This is because these sections of metadata are "derived" metadata: the Schema Generator produces Actuals metadata, and Aggbuilder produces Agg metadata. As a result, a new EpiCenter can be constructed using the Export All metadata by rebuilding the schema, re-extracting, and re-running Aggbuilder.

To "clone" an EpiMeta in such a way that EpiMart itself will not need any modification when it is used with this new EpiMeta, you must select the options for Actuals and Aggs.

> *Note:* An Export All operation on a running system, followed by a re-import of that metadata causes all Actual and Agg metadata to be lost.

Export File Format

Each Microsoft Access Export database has the same schema. In fact, this schema can be thought of as a meta-schema for representing relational data. For a description of these tables, see Table H-1.

To modify the row contents contained in an Export file, edit the *Export_col* table, which is simply a collection of name/value pairs for columns.

Table H-1: Export Tables

Table Name	Description
Export_tbl	One row per metadata table being exported.
Export_row	One row per metadata row being exported.
Export_col	One row per column per row of metadata being exported. Only non-relationship columns are contained in this table.
Export_rel	One row per relationship between two rows of metadata. Can be a relationship between two rows contained in the Export file, or between one row in the Export file and one reference to a row in a foreign EpiMeta.
Export_status	Header information about the Export file.
Rel_parent	A reference to a metadata row in the foreign EpiMeta.

APPENDIX I

Troubleshooting

This chapter describes the Epiphany error conditions and error messages and suggests action you can take to resolve problems.

If you need additional assistance, please e-mail Epiphany Customer Support: support@epiphany.com.

Registry Editor Warning

Using Registry Editor incorrectly can cause serious problems that may require you to reinstall your operating system. Microsoft cannot guarantee that problems resulting from the incorrect use of Registry Editor can be solved. Use Registry Editor at your own risk.

For information about how to edit the Registry, view the "Changing Keys And Values" Help topic in Registry Editor (**Regedit.exe**), or the "Add and Delete Information in the Registry" and "Edit Registry Data" Help topics in **Regedt32.exe**.

Note that you should back up the Registry before you edit it. If you are running Windows NT, you should also update your Emergency Repair Disk (ERD).

SQLServer Error Message

Cannot Connect to the Server

The most likely cause of this problem is that the ODBC driver for SQLServer has not been correctly installed.

You can verify this by opening Start\Settings\Control Panels\ODBC\ and selecting the tab for the ODBC Drivers. Make sure there is an entry for SQLServer. If not, the Microsoft Back Office CD-ROM, Disc 2 CD-ROM, which you used to install SQLServer and its utilities has the ODBC driver in the directory *i386/utils/odbc*.

Application Server Error Messages

The following Application Server errors messages are described:

- *Cannot Connect to the Server* on page 286

- *User Cannot Log In* on page 287

- *Allow Interaction with Desktop Problem* on page 290

- *Out of Memory* on page 290

- *Invalid Object DATE_O Error* on page 291

- *Not a Valid Application Error* on page 292

- *Internal Windows NT Error* on page 292

- *EpiQuery Engine Database Connection Open Failure Exception* on page 293

- *Charts Do Not Display* on page 293

- *GIF Images Fail to Display on Web Pages* on page 297

- *No Results Available for a Query* on page 298

- *Result Page Error: Extraction Date Unknown* on page 298

- *Web Server Message: Object Not Found* on page 299
- *Browser Crashes When Retrieving Results from Application Server* on page 301
- *Refresh Program Fails* on page 301
- *Application Log Full Error* on page 301
- *Application Server Log Security Problem* on page 302

User Cannot Log In

There are three types of failure associated with an unsuccessful login to the Application Server:

1. The Application Server generates a critical exception indicating an application error, or an error related to the NT security domain).

   If this case, an error message appears in the browser with a link to the log that contains the stack trace for the error. Read the error description. Generally, it relates to the way NT domain security is configured or set up at this point. For example, the error might say that the domain controller could not be reached.

   Try to fix the problem based on the specifics of the error message. If you need additional assistance, please e-mail Epiphany Customer Support: support@epiphany.com.

2. An invalid login message displays even though there is a valid username and password.

   This error may result when the search for the username/password occurs on the wrong machine. For example, suppose a user *foo* existed on both the machine local to the Application Server and the primary domain controller.

The order in which the user *foo* is searched is as follows:

- the local SAM
- the primary domain
- the trusted domains

Thus, user *foo* in the local SAM will be found first, even though the username/password combination could have been for the user *foo* that exists in the primary domain or in a trusted domain.

To solve this problem, further specify the username by including the domain name before the username; for example, *EPIPHANY\foo*. In general, specify the full user's name if the user's browser displays an invalid login message.

If the error persists, please e-mail Epiphany Customer Support: support@epiphany.com.

3. Authentication was successful, but the user is not allowed to use the Epiphany system.

For a user to have access to Epiphany System, he or she must be a member of at least one Epiphany group after a sync-up process.

Use EpiCenter Manager to check group memberships after you receive this error message. If the old group memberships were removed, then this error has occurred because in the NT domain, the user is not a member of the NT groups that have the corresponding groups marked Synchronize in EpiCenter Manager.

Also check that the username used to log in matches the username in EpiCenter Manager. If the username in EpiCenter Manager is prefixed with domain name other than the one that the actual NT user is a member of, then the login could fail and return an error message to this effect. Specifying the full username upon log in may fix this problem.

If the error persists, please e-mail Epiphany Customer Support: support@epiphany.com.

Additional Action to Take If User Still Cannot Log In

Note: If you need to have a working system immediately, you may temporarily disable security by hooking up EpiPassThruLogon module. Be sure to read *Application Server Security* on page 186 for instructions on configuring authentication modules.

If you cannot fix the problem using the above procedures, e-mail a security log (*xxx-xxx-xxx-SECURITY.txt*) and Application Server log (*xxx-xxx-xxx-SRV*) to Epiphany Customer Service.

A description of the three security logs follows:

`APPSERVER.LOG`	The only file with a suffix of *.log*, this is the main Application Server activity log, which logs a summary of all queries performed on that server. Includes the user who submitted the query request, the type of ticksheet submitted, the failure or success of the query, and the amount of time for the query.
`SAVE_RESTORE`	Logs all save and restore operations during a session.
`QM_randomly_generated_alphabetic_sequence_username`	Consists of individual logs, one per user, that show the content of each query or other browser transactions. These logs contain the selected attributes, measurement selections, filters, and so forth submitted with each query. It also contains the SQL sequences, the duration times of the various stages of the query, and the total time for each query.

Allow Interaction with Desktop Problem

If you have installed an Application Server and the Service Control Manager (Start Menu\Settings\ControlPanel\Services) fails to start the Application Server, then double-click the service in question, and make sure the Allow Service to Interact with Desktop is selected.

Out of Memory

The Application Server requires sufficient memory (at least 64 Mb) to function properly. The default jre program allows the Application Server to allocate only 16 Mb of extra memory. Queries that involve Product and Customer (or any large dimension) and have the Rows option set to All can easily exhaust the default 16 Mb. When this happens, you will receive an Out of Memory exception error: `java.lang.OutOfMemoryError`.

The solution to this problem is to make sure that you have started the Application Server with the `-mx64M` parameter. For example,

`jre -mx64M -classpath ... com.epiphany.server.Server ...`

The `-mx64M` parameter allows the Application Server to allocate up to 64 Mb of memory.

The Refresh program can also exhaust memory if run three times consecutively. This is because the atomic switch from old to new metadata requires a least two copies of the metadata to be alive at the same time. Until all references to the old metadata are released, both objects stay in memory. Eventually, as sessions are cleaned up and as old command threads die, the old object will be released.

VirtualMartDatabase Key Missing Error

If you receive this message from the Application Server, more than likely a valid EpiCenter data store has not been specified.

When you configure an EpiMeta database using EpiCenter Manager, you will use the EpiMart Data Store dialog box in the Extraction\Data Stores folder to specify the target EpiMart. Click the Properties tab and make sure that the data store name is correct.

Invalid Object DATE_O Error

When starting the Application Server, you may encounter the SQL error `Invalid object Date_0`. This error is generated when the Application Server attempts to read the metadata (EpiMeta).

To correct this problem:

- Make sure that the EpiMart data store specified via EpiCenter Manager points to a valid EpiMart database. (Open the Extraction\Data Stores\EpiMart Data Store dialog box, and click the Properties tab.)

- Make sure that you have populated the date dimension in your EpiCenter using EpiCenter Manager. To verify this, go to ISQL and use the `sp_help` command on your EpiMart database. Check for the existence of the *Date_O* table. If it is not there, it has not been populated. See *Getting Started* on page 59 for instructions.

Not a Valid Application Error

This problem can occur for two reasons:

- If a service component required for Windows NT, an application, or a network protocol, is corrupted or missing.

  To correct this problem, manually expand the service component file. For example, if the *service name* in Event ID 7000 is MUP, expand MUP.SY_ from the Window NT CD-ROM to MUP.SYS in the *%SystemRoot%\SYSTEM32\DRIVERS* folder.

- If the folder location of the executable contains spaces in the directory name (that is, has a long filename). An example would be when the executable is located in the *\Program files\service.exe* folder.

  To correct this problem, modify the Registry key that contains the executable path so that it is enclosed in quotation marks.

Internal Windows NT Error

This error results when the Epiphany Application Server cannot be started as a service. The exact error message that was returned from the Epiphany Application Server is logged in the Windows NT Event Log.

To locate this error message in the Event Log:

1. Go to Start menu\Programs\Administrative Tools (Common)\Event Viewer.

2. Click the Log menu and select the Application.

3. Double-click the appropriate event.

   All Epiphany Application Server events have the source EpiAppServer.

To solve this problem, first stop any running Application Server services. Then log onto Windows NT as an administrator and re-install the Epiphany software. If this does not solve the problem, start the Application Server from the command line as described in *Running as a Console Application* on page 164. The console output should describe what is wrong.

EpiQuery Engine Database Connection Open Failure Exception

An EpiQueryEngineDBConnOpenFailureException from the EpiQueryEngine means that the Application Server is having difficulty connecting to the database. Take these steps to correct this problem:

- Make sure that the username and password for the EpiMeta database are set correctly in the Windows Registry.

- Make sure that the database name and server are also configured properly (again, in the Registry).

- Make sure that SQLServer TCP/IP Sockets have been enabled. Usually, this is option is set when the SQLServer is installed on the machine. To verify this, from the Start menu, open SQLServer 6.5 and select the SQLServer Setup program. Then click Change Network Options and make sure that TCP/IP sockets are enabled. Restart SQLServer if you change any values.

Charts Do Not Display

If there is a broken link to an image, or no image appears, or an image with an error message appears instead of the chart, there is a charting problem. Note the following:

- **makechart.dll** has a log file that logs all calls from browsers to the **isapi.dll**. It is located in the Epiphany system log directory (set by SystemLogDir Registry key) and is called *charts.log*. You can use this file to check whether calls make it through **makechart.dll**. There are two entries for each call, one when the call initiates and another when the call is completed.

- **makechart.dll** has the following syntax for calling it (HTML reference):

```
http://bullwinkle/STROMBOLI/
makechart.dll?makechart?uid=filename.epc&context=STROMB
OLI&chnum=0&chcollection=1&chtype=4&3d=1&rescale=0&widt
h=600&height=400&debug=0
```

where:

- `uid` is the name of the EPC (*.epc*) file that has data to be charted.

- `context` is the instance name.

- `chnum` is the chart number in the EPC file.

- `chcollection` is the collection number in the EPC file.

- `chtype` is the chart type.

- `3d` is the 3dflag.

- `rescale` is the rescale flag.

- `width` is the width of the image to return.

- `height` is the height of the image to return.

- `debug` is the flag used for debugging.

If there is an error during chart creation in **gsmaker.exe**, **makechart.dll** generates an image with an error message in it. If the debug verbosity level is set to 0, the error message is `Charts are not available`. If debug is set to 1, the error message should be more informative. It may describe a problem with an EPC file, or show an exception.

Before attempting to diagnose any charting problem:

1. Try to isolate the part of HTML (the URL) that calls charts, and enter it into the URL address box in the browser window.

2. View the source on the page that includes charts.

3. Find the part of HTML that calls **makechart.dll** and paste it into the URL address box.

4. Change the debug flag from 0 to 1.

5. Go to that Web page.

If there is a broken link to an image or no image appears, the problem could be caused by **makechart.dll** not being called, or **gsmaker.exe** looping or crashing. Possible causes for this problem are:

- **makechart.dll** is not installed correctly. Check the *WWWROOT* directory for the instance for **makechart.dll**. Make sure it is present. See *Manual Chart Install* on page 220 for instructions on installing **makechart.dll**.

- **makechart.dll** has different parameters from the HTTP request actually being made to it. Compare the actual HTML request with the HTML syntax for calling **makechart.dll** given above. If the two are different, make sure that your Application Server and **makechart.dll** derive from the same build of the product.

- **gsmaker.exe** has an application error that put it in an infinite loop or caused it to crash. Check to see if **gsmaker.exe** is consuming all the CPU's resources. If it is, then kill the program, or reboot the machine.

  Check for an existence of **GSW32.exe** process. This is the graphics server package engine. It should go away after a chart is successfully created. If the process is still running, that means a severe failure has occurred. If it seems that **gsmaker.exe** is not functioning properly, e-mail the EPC file used to generate this chart and the **makechart.dll** HTML reference to Epiphany Customer Service.

If an error message is returned instead of a chart, your problem could be the result of one of these circumstances:

If an error is COM (Common Object Model) related, check for the following errors. COM error codes start with 800 and are most likely caused by incorrect or incomplete setup.

- `Not enough storage is available to complete this operation.`

  This error indicates **gsmaker.exe** terminated during creation of the chart. E-mail the EPC file used to generate this chart and the **makechart.dll** HTML reference to Epiphany Customer Service.

- `Access is denied.`

  This error indicates **gsmaker.exe** is not properly installed. Use **dcomcnfg** to determine the special group that Everyone has access to and to launch permissions to the EpiChart Class COM object. See *Manual Chart Install* on page 220 for more information.

- If an error is **gsmaker.exe** related (begins with `GSMaker error`), then check for these errors:

  - If there is an EPC file that is related, then the Application Server created a bad .EPC file. E-mail the EPC file and the **makechart.dll** HTML reference to Epiphany Customer Service.

  - If the error says `could not initialize graphics server` subsystem, check that Graphics Server files have been installed. (The Graphics Server consists of two parts: **makechart.dll**, which should never have any problems and **gsmaker.exe** and its auxiliary files, which could be installed incorrectly or configured poorly by the installation program.) If these files are there, reboot the computer, run the same query and the chart should show up. If this occurs, then **gsmaker.exe** is slowly leaking resources. Report this problem to Epiphany Customer Service.

    If no chart is returned and a dialog box on the Application Server machine that indicates the Initialization of the dynamic linked library **user32.dll** failed, and the process is terminating abnormally, then **gsmaker.exe** is not properly installed.

Use **dcomcnfg** to determine if EpiChart Class is running under the Identity of the query user. See *Manual Chart Install* on page 220 for more information.

If the problem persists, e-mail Epiphany Customer Service.

GIF Images Fail to Display on Web Pages

If GIF images do not appear on your Web pages, do the following:

1. Make sure that your Web server has an alias for your *instance_name* that points to a valid directory. If you performed a normal installation, then all Web files should be located in the directory:

    C:\Program Files\Epiphany\Instance_name\web\WWWROOT

    and GIF files should be located in the directory:

    C:\Program Files\Epiphany\Instance_name\web\WWWROOT\images

    (assuming your Epiphany Application Server was installed in *C:\Program Files*).

    If your Web server is IIS 3.0, you can find the aliases by opening up the IIS Internet Service Manager (normally this is located in your *Start\Microsoft Internet Information Server* menu) and selecting the Directory tab. Make sure that there is an entry for *instance_name* that has a valid directory.

2. Make sure the *WWWROOT\images* directory has the GIFs in it. If you are missing a few GIFs, then you will need to reinstall your Application Server or copy the GIFs from a different instance.

3. Check the BASE HREF tag that is defined in the source of the page. In your browser, try to view the source for this HTML page. Look for the BASE HREF tag at the top of the page. Note what it is and make sure that it is a valid alias using the procedure above.

4. Make sure your Web server is serving pages and that your browser is not displaying cached HTML. Clear the caches (Memory and Disk) on your browser, close your browser, and try to access the URL again. Also, try referencing another URL from your Web server to make sure that it is running.

   Technical Note: All of the GIFs on an Application Server-generated page are referenced from the *images* directory, which is relative to the BASE HREF specified at the top of the page in a META tag. The *instance_name* is derived from the URL that you use to access the Application Server. It is used throughout the system to read the correct Registry entries and to generate the correct URLs.

No Results Available for a Query

The most common cause of this problem is that the *current_datamart* key in the EpiMeta database is set to the wrong database.

   Use EpiCenter Manager to make sure that the *current_datamart* is set to the correct database. You can also use the ISQL tool to determine which database, A or B, has the most current data.

Result Page Error: Extraction Date Unknown

The last_extract_date is a field that is kept in the EpiMeta database. It is used to keep track of the date displayed on the top of all Clarity and Relevance reports as the date of the last extraction. It is normally populated by the extraction SQL entered in the End of Extraction job in the EpiCenter Manager. It can also be populated by EpiCenter Manager via the Configuration dialog box. This field must be entered in one of two very strict formats. The default format is mm/dd/yyyy; for example: 01/14/1998.

The Application Server applies the following logic to parse the date:

1. If the field has more than 10 characters, then parse it using the pattern mm/dd/yyyy hh:mm:ss. Otherwise, use the pattern mm/dd/yyyy.

2. If the parse fails, then use {extraction date unknown}.

In addition, the date that is displayed at the top of the report always has a time zone. The time zone is printed based on the default time zone of the machine on which the Application Server is running. The date that is being displayed is also taken into consideration. For example, the date 12/20/1997 will display as December 20, 1997 PST if the Application Server was running on a machine in California. However, the day 05/12/1998 will display as May 12, 1998 PDT on the same machine since Daylight Savings time took affect in April.

> *Note*: EpiCenter Manager does not allow the user to enter a date in the format mm/dd/yyyy hh:mm:ss. Only the SQL in the extraction job can enter dates of this format, or you can manually arrange this via ISQL.

Web Server Message: Object Not Found

If you installed the Epiphany software using the standard Epiphany software installation program, you will access your Web server through this type of URL: http://machinename/scripts/*instance_name*/Epiphany.dll.

If you receive an object not found error message, follow these steps:

1. Start and stop the Web server. If this does not solve the problem, go to the next step.

2. Verify the Web server is serving pages.

   Note: Make sure that your browser is not just returning cached HTML pages by clearing your memory and disk cache before testing.

   Try to access other URLs from the same *machinename*. Try to access other static HTML files that are installed as a part of the Application Server installation, such as http://machinename/*instance_name*/clarityhelp.html.

3. If this does not work, try accessing any other file that the Web server should be serving. Consult the Internet Service Manager for the names of other aliases that the Web server should be serving, and then try to access these aliases with your browser.

In most cases, your Web server searches the
C:\Inetpub\scripts\instance_name directory to find the **Epiphany.dll**
program. Make sure that there is such a directory on your machine, and
that the **Epiphany.dll** file is in that directory.

4. Check the file permissions for the **Epiphany.dll** file.

   First, make sure that the account IIS uses for anonymous logins has file
   access permissions for the **Epiphany.dll** file. Go to the IIS 3.0 Internet
   Service Manager and look at the Anonymous Login account box. In IIS
   4.0, right-click the name of the machine and choose Properties. Select the
   Directory Security tab. In the Authentication Methods dialog box, select
   Allow Anonymous Access and click the Edit button to modify the account
   used for this purpose. Make sure the user and password are correct.
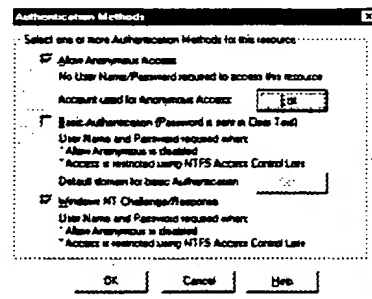
Figure H-1 Authentication Methods Dialog Box

To check file permissions, open the Windows NT Explorer window and
right-click the **Epiphany.dll** file in the *./scripts/instance_name* directory
specified above. Go to Properties and open the Security tab. Click View
Permissions and make sure that the account that you used for
Anonymous Web Login has access to this file. (If Everyone is selected,
then all people, including the Web login account, have access to the file.)

Browser Crashes When Retrieving Results from Application Server

In general, for machines with less than 32 Mb of RAM, the browser will perform very poorly when parsing HTML files that are larger than 150 K. Therefore, users who do not have at lease 32 Mb of RAM installed on their machines sold refrain from retrieving large queries.

An example of a large query follows. Suppose you query Customer by Fiscal Year and apply the filter Business Unit: Learning. This query returns approximately 3800 customers, and the HTML that is generated is 1.8 Mb. When loaded in Netscape 4.03, it occupies 37 Mb, takes 3 minutes to parse, and consumes the entire processor. The parsing is the bottleneck in the downloading of the file.

> Note: When started, Netscape Navigator 4.03 requires 8 Mb immediately to load whatever it is loading. Internet Explorer (IE) requires 6.7 Mb, and is substantially faster at parsing the text.

Refresh Program Fails

If the Refresh program fails, the Application Server will continue to use the old metadata information. Users will still be able to log in and view the old ticksheets. This is because the Refresh program reads the new metadata into temporary structures that are atomically exchanged with the old structures only if all of the refresh structures were read correctly.

To solve this problem, rerun the Refresh program.

Application Log Full Error

If you receive this error, you can take the following action to delete all log events.

> *Note:* If you do not want to delete all of the log events, you can also increase the amount of space allocated to the Application Log. You can also select the Override as Needed option in the Log Properties dialog box.

1. Open theStart\Programs\Administrative Tools (Common)\Event Viewer. Select the Application Log under the Log menu item.

2. Select Clear All Events from the Log menu.

   When prompted whether you want to save the log files, and whether you really want to delete all of the log events, respond in the affirmative.

Application Server Log Security Problem

The *APPSERVER* and *SRV* log contain all of the session IDs that have logged in. Although unlikely, it is possible that someone could alter a session ID (along with a fake IP address) and run queries against the Application Server.

The log file also contains information about who is running a query, and when they are running a query, which may be confidential information.

The log files for each query contain all the information necessary to rerun a query. Hence, a malicious party can also determine the type of queries run by a particular user.

It is important to give user access to the log file directory so that when problems occur, the Download query log link at the bottom of each results page will work. Directory browsing for the log file directory, however, should be turned off.

To turn off directory browsing, de-select the Directory Browsing Allowed option in the Directories tab of the WWW Services Properties dialog box in the Internet Service Manager. (Normally, IIS is located in your *Start\Microsoft Internet Information Server* menu.)

GLOSSARY

Actual Tables

The Actual tables are metadata created by the Adaptive Schema Generator after it has built or modified tables in EpiMart. The Actual tables are consulted on subsequent schema generations to determine delta operations to perform. Also, other tools examine these tables as a check to ensure that EpiMart's schema matches the current metadata definition.

Adaptive Schema Generator

The Adaptive Schema Generator is a component of EpiCenter Manager that builds the tables in EpiMart using the schema metadata definitions in EpiMeta. When schema metadata changes, this program can perform delta operations to modify the schema without losing data in EpiMart.

Aggregate Builder (Aggbuilder)

Aggbuilder is the executable program (**agg.exe**) that builds aggregate tables in EpiMart. Normally. the execution of this program is scheduled after all data has been extracted from source systems and merged into EpiMart.

Aggregate Group

An aggregate group is a metadata definition of the fact aggregates to be built. It is a series of instructions that tell Aggbuilder which aggregates to build on one or more fact tables. An aggregate group applies to a single constellation. The actual fact aggregates that get built are determined by a combinatorial expansion of the instructions in the aggregate group.

Aggregate Navigation

Aggregate navigation is the process by which the Epiphany query machinery determines the optimal aggregate table to use to satisfy an application's request for information. Applications such as Clarity do not need to know about the presence of aggregates because the aggregate navigation process makes the aggregate layer abstract.

Application Server

The Epiphany Application Server is the Windows NT Service that connects with a Web Server and serves up Epiphany ticksheets and reports. An Application Server is configured to connect with an EpiCenter.

Attribute

An attribute is a dimension selection on a ticksheet. The row and column lists in Clarity contain attributes. Attributes are related to exactly one dimension column in a table. The only difference between an attribute in a table and one in the Epiphany system is that an Epiphany attribute has an associated display label, such as Fiscal Year, that can be configured via Web Builder.

Attribute Role

An attribute role is the usage of an attribute on a ticksheet. Attributes are defined only once per ticksheet. If you want a single attribute to appear in both the rows and columns listboxes of Clarity, use Web Builder to assign that attribute both of these roles: Clarity Row and Clarity Column.

Backlog Type

Backlog type is used on a measure term to specify that the measure term should exhibit accumulation behavior. When time is one of the dimensions of a query, the results for different time periods will exhibit beginning or ending accumulated values instead of the actual transactions that occur in that time period. BEGIN specifies beginning accumulated value, whereas END specifies ending accumulated value.

Base Dimension

A base dimension is a physical dimension table in EpiMart, such as Customer and Product. Base dimension tables contain the actual dimensional values that are available for reporting or filtering. Base dimension tables can be used more than once in a single constellation via dimension roles. Base dimensions are defined for the entire EpiCenter and can be shared by all of the constellations within the EpiCenter.

Base Dimension Aggregate

A base dimension aggregate is a physical table in EpiMart that represents an aggregated view of a base dimension base table. A base dimension aggregate results from the removal of one or more columns from the base table, followed by the removal of duplicate rows caused by this deletion.

Base Table

Base tables are the EpiMart tables (both fact and dimension) that store non-aggregated customer data at the level of granularity of extraction. These tables will be used as input to Aggbuilder to build aggregate tables with the same information at different levels of granularity. Base table names end with an underscore zero (_0); for example, Order_0_A or Order_0_B.

Constellation

A constellation is a grouping mechanism for like-structured fact tables. A constellation defines the dimensionality of all fact tables in that constellation. Dimension roles and degenerate dimensions are defined on a constellation, not a specific fact table. All fact tables placed in that constellation then inherit that definition. Both ticksheets and measures are defined within the scope of a single constellation, and thus apply only to the dimensionality defined by that constellation.

Custom Index

A custom index is the metadata definition of indexes to build on fact tables in EpiMart. Normally, each fact table is given a single index. You may use EpiCenter Manager to specify additional indexes to build. You must use the semantic template Custom Fact Index to actually build the indexes.

Data Store

A data store is a logical location of data to be used either as a source or sink within an EpiCenter. Data stores include relational database connections, and also files and directories.

Dataset

A dataset is a logical user-interface grouping of ticksheets that display the same view of data. Typically, different ticksheets types (such as Clarity and Relevance Profiling) that show the same data within a constellation are placed within a dataset. The dataset appears to end-users as a named entity that corresponds to a single business process. For instance, a dataset called *Executive Summary* might be defined that shows similar amounts of high-level data, using different Epiphany applications.

Date Dimension

The Date dimension is a special base dimension table supplied by Epiphany, which is used for storing all attributes related to time. All fact tables in an EpiCenter receive the foreign key *date_key* to this table.

Degenerate Dimension

A degenerate dimension is a column in a fact table that stores textual information in lieu of using a foreign key to a base dimension table. Degenerate dimensions are used when a single field of data fully specifies a dimension of that fact. Examples include Invoice Number and Serial Number.

Because degenerate dimensions appear only in fact base tables, they are never aggregated. (They are, however, always indexed.) Degenerate dimensions are defined on a constellation rather than for a specific fact table. All fact tables within that constellation inherit the degenerate dimension definition.

Dictionary Entry

A dictionary entry provides end users with online definitions of the terminology used on a ticksheet, such as Units, Gross, and Sell-Through, which are hyperlinked to a dictionary page. The creator of the ticksheet configures these entries via Web Builder.

Dimension Column

A dimension column is a single column or attribute of a base dimension table. It contains the actual customer data that appears on reports or in filters.

Dimension Column Access

Dimension column access is the ability for a user or group to view only a subset of the available data in an EpiCenter. For example, dimension column access settings can be used to limit specific users to only data for the Western Region.

Dimension Column Set

A dimension column set is a named set of dimension columns (all within a single base dimension) that defines which columns will be used in a base dimension aggregate.

Dimension Role

Dimension roles allow a single base dimension table to be used in different roles within a constellation; for example, a Territory base dimension table includes Eastern Sales and Western Sales data. A Western Sales dimension role would reference the Territory base dimension for its data.

A dimension role is a dimension column in a fact table that defines the foreign key that references a base dimension table. It always has an associated base dimension table. Dimension roles are defined within a constellation, and all fact tables in that constellation inherit the dimension role. Multiple roles within a single constellation can refer to the same base dimension.

EpiCenter

An EpiCenter is a single logical Epiphany database installation that includes customer data in addition to control metadata. It physically consists of two databases: EpiMeta and its associated EpiMart.

EpiCenter Enterprise Manager (EpiCenter Manager)

EpiCenter Enterprise Manager is a Microsoft Windows program for configuring EpiCenters. These key components of the Epiphany system are available via EpiCenter Manager: schema, extraction, and security metadata, the Adaptive Schema Generator, and metadata export/import.

EpiChannel

EpiChannel is the Epiphany program (**extract**) that executes extraction jobs. The person who runs EpiChannel enters parameters to connect to an EpiCenter and specifies jobs to be executed. EpiChannel then implements the extraction steps declared as metadata in the EpiCenter.

EpiMart

The physical database that contains actual customer data. The schema of tables in EpiMart is determined by the metadata in EpiMeta. The data contents of EpiMart tables are determined by the extraction steps in EpiMeta.

EpiMeta

EpiMeta is the metadata repository for an EpiCenter. It contains all control information for the EpiCenter and therefore defines the behavior of that EpiCenter. EpiMeta points to EpiMart via the special EpiMart data store, which is available in every EpiMeta.

External Column

An external column is a single column in an external table.

External Table

An external table is a table built in EpiMart, usually as a temporary table used during extraction. External tables must be declared as metadata (as opposed to begin built outside of Epiphany) in order for the Adaptive Schema Generator to know of their existence. Otherwise, the table would be purged by the Purge EpiMart command.

Extraction Group

An extraction group is a set of extraction steps.

Extraction Step

An extraction step is a single atomic extraction operation. It can be either a SQL statement or a semantic Instance.

Extractor

An extractor is a list of extractor steps together with input and output data stores. It logically specifies a series of steps that move data from the input to the output data store.

Extractor Step

An extractor step is a single step of an extractor, which always points to an extraction group. In this way, extractors consist of an ordered list of extraction groups (which are ordered lists of extraction steps).

Fact Aggregate

A fact aggregate is a physical table in EpiMart that contains aggregated fact information for a single fact table.

Fact Column

A fact column is a single numeric column in a fact table.

Fact Table

A fact table is a physical table in EpiMart that contains numeric data in addition to references to dimension tables that specify attributes about the fact, such as who, what, when, and where the fact occurred.

Filter Block

On a ticksheet, a filter block controls the user's ability to filter on a single dimension column (for a dimension role of the ticksheet's constellation). The filter block also defines the appearance of the filter (for example, check box versus listbox).

Filter Element

A filter element is a single check box for filtering within a filter group, or a single entry within a listbox filter block.

Filter Group

A filter group is a logical grouping of filter elements within a filter block. It is applicable only to check box filter blocks.

Job

A job is a top-level workflow object that defines a sequence of steps to be performed by EpiChannel. It also specifies the logging locations for that execution.

Job Step

A job step is a single step of a job, which can be either an extractor or a system call.

Measure

A measure is a single business calculation. It can be an arithmetic combination of measure terms using RPN (Reverse Polish Notation).

Measure Mapping

The mapping of ticksheet selection columns to measures.

Measure Term

A measure term is a single component of a measure. It refers to the aggregation of a single fact column, such as *SUM(Order.net_price)*, with a particular transaction type. It can be combined with other measure terms to create a composite measure (business calculation).

Pull/Push SQL Statement

A pull/push SQL statement is a type of extraction step that issues SQL against an input data store and then pushes the result set into a table in the output data store of an extractor.

Query Machinery

The component of the Epiphany Application Server that communicates with EpiMart and actually issues SQL statements against the DBMS. The query machinery is a common component of Epiphany's front-end applications.

Report Gallery

The Report Gallery is a component of Epiphany's front-end user interface that allows users to view Saved Queries.

Saved Query

A Saved Query is a saved set of ticksheet settings from a previous query that can be viewed again. Users and groups are granted permission to use or modify Saved Queries. Users who request the same Saved Query (and have the same dimension column Access rights) will receive the same output (reports, charts, and so forth).

Selection Column

A selection column is a single column in the measure section of a ticksheet. The combination of one value from each selection column translates into a single measure in the report output.

Selection Column Element

A selection column element is a single value in a selection column. For instance, a selection column might contain the elements *Gross, Net,* and *Return.*

Semantic Instance

A semantic instance is a post-compilation SQL program. When a semantic template is applied to either a base dimension or fact table, the template becomes instantiated as an actual SQL program that can be used to accomplish specific business rules during extraction.

Semantic Template

A semantic template is a generic SQL program intended to accomplish specific business rules (these rules are referred to as semantic types) during extraction. A semantic template does not refer to actual column or table names. Only when the template is applied to a base dimension or fact table (via a semantic instance) does the SQL contained within it refer to real column and table names.

Semantic Type

A semantic type refers to the logical business process for which the semantic template is applied.

Source System

A source system is the logical notion of a source of business data. Two physical databases (one a backup of the other) might represent the same source system within Epiphany.

SQL Statement

A SQL statement is an extraction step that issues custom SQL against an input data store. The results of the SQL statement can either be discarded or used to push data into a table in the output data store.

Stand-Alone SQL Statement

A stand-alone SQL statement is a SQL statement whose results are discarded.

System Call

A system call is an extraction step that causes an operating system program to be invoked.

Ticksheet

A ticksheet is a form that end users open in a Web browser and use to submit queries to the data warehouse. It is called a ticksheet because users make selections by ticking (selecting) items.

The ticksheet developer uses the Ticksheet dialog box in Web Builder to construct these ticksheets.

Transaction Type

Transaction types distinguish rows with different interpretations in the same fact table. For example, an Order fact table might contain both a BOOK and SHIP transaction type, differentiated by the value of the column *transtype_key*.

Measure terms created via Web Builder specify a transaction type in addition to a fact column. This allows the summation of only those rows in a fact table with the same transaction type.

Web Builder

Web Builder is a Windows-based program developed by Epiphany for configuring user-interface metadata, including measures, ticksheets, and dictionary entries.

DATE DIMENSION FIELDS

Table 25 describes the date dimension fields that the E.piphany system uses.

TABLE 25: DATE DIMENSION FIELDS   (1 OF 4)

dim_col_name	Description
cq_and_cy_name	Calendar quarter and year name. For example, Q1 1999.
cq_name	Calendar quarter name. For example, Q1.
cy_name	Calendar year name. For example, 1999.
date_key	Primary key—date as a native date type.
day_cq_begin	Whether or not this is a day on which a calendar quarter begins. (1/0).
day_cq_end	Whether or not this is a day on which a calendar quarter ends. (1/0).
day_cy_begin	Whether or not this is a day on which a calendar year begins. (1/0).
day_cy_end	Whether or not this is a day on which a calendar year ends. (1/0).
day_fq_begin	Whether or not this is a day on which a fiscal quarter begins. (1/0).
day_fq_end	Whether or not this is a day on which a fiscal quarter ends. (1/0).
day_fy_begin	Whether or not this is a day on which a fiscal year begins. (1/0).

TABLE 25: DATE DIMENSION FIELDS

dim_col_name	Description
day_fy_end	Whether or not this is a day on which a fiscal year ends. (1/0).
day_month_begin	Whether or not this is a day on which a month or period begins. (1/0).
day_month_end	Whether or not this is a day on which a month or period ends. (1/0).
day_name	The day as a native date type.
day_name_char	Date (as a string). For example, Apr 02 1995.
day_name_char_weekday	Date (as a string) with weekday prefix. For example, Sun Apr 02 1995.
day_number_in_cq	The number of this day in the calendar quarter, starting at 1.
day_number_in_cy	The number of this day in the calendar year, starting at 1.
day_number_in_fq	The number of this day in the fiscal quarter, starting at 1.
day_number_in_fy	The number of this day in the fiscal year, starting at 1.
day_number_in_month	The number of this day in the month or period, starting at 1.
day_number_in_week	The number of this day in the week, starting at 1.
day_number_til_end_cq	The number of days until the end of the calendar quarter, ending with 1.
day_number_til_end_fq	The number of days until the end of the fiscal quarter, ending with 1.
fq_and_fy_name	Fiscal quarter and year name. For example, Q1 1999.
fq_name	Fiscal quarter name. For example, Q1.
fy_name	Fiscal year name. For example, 1999.
month_and_cy_name	The name of the month abbreviated and the calendar year. For example, Apr 1999.

TABLE 25: DATE DIMENSION FIELDS

dim_col_name	Description
month_and_fy_name	Month (abbreviated) or period name and fiscal year. For example, Apr 1999.
month_name	The three-letter abbreviations of the month (without a year or any other value). For example, Apr.
month_number	Month or period number since the first date in the system, starting at 1.
month_number_in_cq	Month number since the start of the calendar quarter, starting at 1.
month_number_in_cy	Month number since the start of the calendar year, starting at 1.
month_number_in_fq	Month or period number since the start of the fiscal quarter, starting at 1.
month_number_in_fy	Month or period number since the start of the fiscal year, starting at 1.
month_number_til_end_cy	Number of months or periods until the end of the calendar year, ending with 1.
month_number_til_end_fy	Number of months or periods until the end of the fiscal year, ending with 1.
vacation_day	Whether or not this day is a vacation, (1/0).
week_friday	Date (as a string) for the Friday of the current week. For example, Apr 01 1994.
week_monday	Date (as a string) for the Monday of the current week. For example, Apr 01 1996.
week_number	Week number since the first date in the system, starting at 1.
week_number_cq	Week number since the start of the calendar quarter, starting at 1.

TABLE 25:   DATE DIMENSION FIELDS

dim_col_name	Description
week_number_cy	Week number since the start of the calendar year, starting at 1.
week_number_fq	Week number since the start of the fiscal quarter, starting at 1.
week_number_fy	Week number since the start of the fiscal year, starting at 1.
week_number_til_end_cq	Week number until the end of the calendar quarter, ending with 1.
week_number_til_end_cy	Number of weeks until the end of the calendar year, ending with 1.
week_number_til_end_fq	Week number until the end of the fiscal quarter, ending with 1.
week_number_til_end_fy	Number of weeks until the end of the fiscal year, ending with 1.
week_saturday	Date (as a string) for the Saturday of the current week. For example, Apr 01 1995.
week_sunday	Date (as a string) for the Sunday of the current week. For example, Apr 02 1995.
weekday	Weekday prefix; for example, Sun.
month_name	The three-letter abbreviations of the month (without any year or any other value).